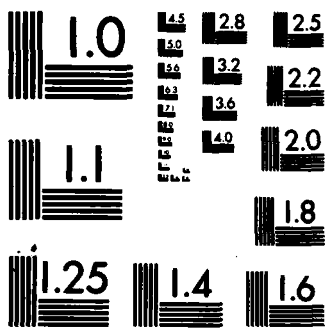
AD-A140 295 SIMULATION EXPERIMENTS WITH GOAL-SEEKINN ADAPTIVE ELEMENTS(U) MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INNORMATION SCIENCE A G BARTO FEB 84 1/2

UNCLASSIFIED AFWAL-TR-84-1022 F33615-80-C-1088 F/G 6/4 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

12

AFWAL-TR-84-1022

SIMULATION EXPERIMENTS WITH GOAL-SEEKING ADAPTIVE ELEMENTS

Andrew G. Barto, Editor
Computer and Information Science
University of Massachusetts
Amherst, MA 01003

February 1984

FINAL REPORT for period June 1980 - August 1983

Approved for public release; distribution unlimited.

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

[illegible]
A. HARRY KLOPF
PROJECT ENGINEER

Donald L. Moon
DONALD L. MOON, CHIEF
INFORMATION PROCESSING TECH BRANCH
AVIONICS LABORATORY

FOR THE COMMANDER

Raymond D. Bellem
RAYMOND D. BELLEM, LT COL, USAF
Deputy Chief
System Avionics Division
Avionics Laboratory

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-84-1022	2. GOVT ACCESSION NO. A140 295	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SIMULATION EXPERIMENTS WITH GOAL-SEEKING ADAPTIVE ELEMENTS		5. TYPE OF REPORT & PERIOD COVERED Final Report for Period Jun 80 - Aug 83
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Andrew G. Barto, Editor		8. CONTRACT OR GRANT NUMBER(s) F33615-80-C-1088
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer and Information Science Department University of Massachusetts Amherst, MA 01003		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F/2312/R1/03
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory (AFWAL/AAAT) Air Force Wright Aeronautical Laboratories (AFSC) Wright-Patterson AFB, OH 45433		12. REPORT DATE February 1984
		13. NUMBER OF PAGES 163
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Adaptation	Neural networks
Adaptive networks	Self-organization
Learning	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes results obtained from computer simulation experiments designed to systematically develop and evaluate an approach to learning by networks of neuronlike adaptive elements. The characterizing feature of this approach is that the network components, or adaptive elements, are self-interested, goal-seeking agents that implement robust algorithms for furthering their individual interests. These algorithms combine stochastic

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Continued

search and associative learning methods. Results of simulation experiments are presented that show how these adaptive elements can cooperate as components of layered networks that adaptively create new features by combining existing features. Other simulation experiments are described that systematically examine the performance of individual adaptive elements in tasks that resemble the tasks faced by elements embeded within networks. The performances of a variety of algorithms are compared and contrasted. These results demonstrate the shortcomings of certain algorithms and the effectiveness of others. Several sets of experiments examine the effects on learning of delayed reinforcement. We justify ~~an algorithm, called~~ the adaptive heuristic critic algorithm, for reducing the severity of the temporal credit-assignment problem for tasks with delayed reinforcement. We illustrate the effectiveness of this algorithm in a task requiring the system to learn to control an unstable dynamical system under the influence of low-quality evaluative feedback.

PREFACE

This report describes a research effort conducted by personnel of the Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, supported by the Air Force Office of Scientific Research and the Avionics Laboratory (Air Force Wright Aeronautical Laboratories) through contract F33615-80-C-1088, "Decision Making by Adaptive Networks of Goal Seeking Components," under Project 2312, Task 2312R1, and Work Unit 2312R103. The research reported here was performed during the period 15 June 1980 to 31 August 1983. The Principal Investigator was D. N. Spinelli and the Co-Principal Investigator was A. Barto. This report was released by its editor, A. Barto, in January 1984.

Since the first vertebrates appeared on the face of the earth, 400 million years ago, they have conquered all habitats. Among the vertebrate species, homo sapiens is dominant. This has come about, without question, because of the human brain. During the last eyeblink of evolution, the human brain has created civilisation as we know it. Vertebrate brains, in general, and the human brain in particular, are machines of immense complexity, capable of adaptation and goal seeking in a great many habitats. Further, most vertebrate brains also possess extremely sophisticated remote sensing systems such as vision and hearing. During the last fraction of the last evolutionary eyeblink, the human brain has created the digital, sequential, computer whose development and use in the last years has been nothing short of astonishing. This machine derives its awesome power from the fact that it is possible to make it perform with high speed and precision "some" of the functions that the human brain performs even though its architecture is totally unlike that of the brain. The successes of Artificial Intelligence in the field of Expert Systems and the difficulties it has encountered in the fields of image understanding and speech recognition seem to indicate that the digital computer, that is, the von Neumann architecture, is particularly good when it comes to cognition and particularly bad when it comes to remote sensing, goal seeking, adaptation and decision making, where brains excel. In a way, if we could simply copy brain architecture onto silicon we would make a giant step forward, just because of the gain in speed (several orders of magnitude). This might in fact become a possibility as neuroscience is making impressive progress in understanding structure and function of systems devoted to adaptation and sensing — a field to which we are ourselves contributing.

Brains are made of neurons and we are beginning to see that neurons themselves are adaptive and goal seeking. This report deals with a number of studies, theoretical and experimental, of goal-seeking adaptive networks composed of goal-seeking adaptive elements. These elements, motivated by a proposal by H. Klopf that neurons are self-interested agents, have been studied by A. Barto and R. Sutton. A variety of adaptive networks of these elements were studied in highly constrained test-bed situations by Barto, Sutton and Anderson. The results here reported show that the networks investigated, even though quite simple, exhibit exciting capabilities when tested on problems that involve classical difficulties such as the temporal credit-assignment problem. Though much remains to be investigated, we believe that networks of robust, self-interested, adaptive elements are prime candidates for non-von Neumann architectures.

D. N. SPINELLI

Principal Investigator

University of Massachusetts

January 1984

ACKNOWLEDGEMENTS

Several people were directly responsible for writing this report and for conducting the research it describes. Chuck Anderson performed the simulation experiments described in Section 2 and wrote particular subsections as part of his Masters' thesis, including the description of the last two simulation experiments with layered networks and the review of previous layered-network research. Rich Sutton designed and performed the simulation experiments described in Sections 3,4,5, and 6 and wrote these sections as parts of his Ph.D. thesis. He also developed the theoretical perspective and analysis provided in these sections. The pole-balancing experiment described in Section 6 was conducted by Chuck Anderson and Rich Sutton using software written largely by Chuck Anderson. I wrote the remaining sections (the introductory and summary sections) parts of Section 2, the description of the pole-balancing experiments in Section 6, and extensively edited all of the other sections. Much additional material, including many simulation experiments and their results, can be found in Anderson's Masters' thesis (Anderson, 1982) and Sutton's Ph.D. thesis (Sutton, forthcoming).

In addition to those directly responsible for producing this report, numerous others contributed to this research effort by encouraging us, allowing us to benefit from their scholarship, and by making this effort possible in the first place. Harry Klopf provided us with the very provacative concept of self-interested, goal-seeking network components. We have found this to be a surprisingly novel, thought provoking, and fruitful idea. Nico Spinelli, Principal Investigator of this project, gave us much appreciated freedom to pursue avenues that appeared promising, provided invaluable insight and guidance at each step, and helped us appreciate some of the subleties of real neural networks. Michael Arbib critically evaluated the theses on which this report is based and was instrumental in providing numerous valuable opportunities to make contact with other researchers having interests similar to our own.

P. Anandan contributed his mathematical talents to several aspects of this research, Steve Epstein permitted us to benefit from his extensive knowledge of the relevant literature, Daryl Lawton pointed us in many useful directions, Lenny Wesley convinced us to use a very powerful text processing system and showed us how to use it, and Susan Parker applied her considerable managerial skills throughout this project. I thank all of these people very much. I also thank all of the "connectionists" in the research community who have argued for the utility of this type of research.

ANDREW G. BARTO

Amherst, Massachusetts

November 1983

CONTENTS

Section 1
INTRODUCTION

This report describes some of the results obtained from computational studies of goal-seeking systems composed of goal-seeking components. Whereas our earlier studies were largely exploratory in nature (Barto and Sutton, 1981b), the studies described here were systematic attempts to understand and to improve the performance of a specific type of adaptive component and to illustrate the behavior of networks of these components. This type of component was motivated by the hypothesis of Klopf (1972, 1982) that neurons are "self-interested" agents that implement behavioral strategies for improving performance determined by their preferences for certain types of inputs over others. We continue to find this a fruitful hypothesis. The study of networks of this type of adaptive component involves many of the issues that arise in the study of the collective behavior of self-interested agents, whether in the context of game theory, economics, or evolutionary biology. We think that this approach has the potential for making adaptive networks useful parts of sophisticated adaptive problem-solving systems.

The results reported here illustrate some of the capabilities of these networks, but they are largely concerned with the systematic development of suitable adaptive elements. It is our contention that to move beyond simple single-layer adaptive networks it is necessary for each network component to implement a fairly sophisticated learning algorithm. An adaptive component that is deeply embedded within a network faces a learning problem that is difficult because of the high degree of uncertainty involved in evaluating its individual performance. Our initial simulation experiments with layered networks (Section 2) convinced us that we were studying the right kind of element, but these experiments also indicated that the elements needed more development. Consequently, many of the results do not involve complicated networks. They may be regarded either as results about single elements or about abstract learning algorithms. However, all of the experiments were motivated by an interest in networks, and all of the single-element learning tasks were designed to reflect aspects of the tasks faced by network components.

Associative Reinforcement Learning

The learning algorithms we have studied are specific types of associative reinforcement learning algorithms. On the surface, associative reinforcement learning is a very simple idea clearly expressed in Thorndike's (1911) famous "Law of Effect":

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. (p. 244)

This describes *associative* reinforcement learning because it emphasizes that associations between stimuli and responses are formed. Although psychologists have demonstrated that the Law of Effect does not provide an accurate account of animal learning behavior (due in part to its symmetrical view of reinforcement and punishment), the general principle it embodies has retained its importance. From the perspective of Artificial Intelligence (AI), this principle has been criticized because it does not suffice to explain how complicated sequences of complicated actions can be learned in reasonable amounts of time. Minsky (1961), for example, has emphasized that in complex learning tasks it is very difficult to "assign credit" for reinforcement to specific decisions made in the course of behavior. Assigning credit merely on the basis of "recency" as suggested by the Law of Effect is not adequate. This well-known difficulty has been largely responsible for the lack of interest in reinforcement learning among AI researchers. It is hard to see how reinforcement learning can be extended to the difficult learning problems whose solutions would have practical importance.

If the results reported here have one central theme, it is that reinforcement learning, especially associative reinforcement learning, involves subtle difficulties that, to the best of our knowledge, have not been elucidated by earlier research. We think that these difficulties bear directly on the problem of extending reinforcement learning to difficult learning tasks. We agree that credit assignment is of critical importance in reinforcement learning, and some of our results bear directly upon credit-assignment problems, but we have also studied phenomena appearing even in very simp.e problems that render common reinforcement-learning algorithms ineffective. For example, we show that certain properties of tasks cause intuitively appealing algorithms always to learn to perform the wrong actions. We think algorithms need to ovecome such difficulties by incorporating mechanisms at *as low a level as possible*. Consequently, we have given considerable attention to the detailed design of low-level learning algorithms that are effective performers across a wide range of conditions. In this way we hope to prevent an attempt to solve a complex learning problem by a complex system from failing due to hidden low-level difficulties. We suspect that the poor performance of some earlier reinforcement-learning systems was due to such hidden problems rather than to mistakes in overall principles. In reinforcement learning, as in other domains, it is quite possible for efforts to be on the right track yet fail due to inadequate understanding of subtleties and inadequate fine-tuning. We believe this to have been the case with computational approaches to reinforcement learning, especially those involving adaptive networks of neuronlike elements.

Methodology

The methodology we used strongly relies on computer simulation. Although we do not regard simulation results as substitutes for mathematical theories, we did not wish to let mathematical convenience dictate the class of learning systems to be studied. Where possible, we attempted to prove mathematical results, but these apply only to restricted cases. A powerful and general mathematical theory may exist, but we have not yet found it. Consequently, we performed empirical analyses of the performance of a variety of learning algorithms by conducting carefully designed simulation experiments. In most cases, these simulation experiments were *comparative* studies that allowed us to determine the relative performance of a number of algorithms in specific tasks. The results of these simulation experiments provide an indication as to what form a mathematical theory should take and what results it should permit one to prove. Additionally, wherever possible, we included algorithms proposed by other researchers in these comparative experiments. In this way we tried to demonstrate advances over existing methods, although the possibility clearly exists that we are not yet aware of all of the relevant existing algorithms.

Great care was taken in designing the tasks used to test and compare learning algorithms. These tasks, or test-beds, are of roughly two kinds. Test-beds of one type involve relatively simple, abstract tasks that test one, and only one, feature of a learning algorithm. By using such abstract test-beds, one is able to determine the features of an algorithm that are responsible for specific features of its performance. Experiments with these test-beds are similar to those conducted by psychologists using animal subjects, including the use of control subjects and tests of statistical significance. We used abstract test-beds for obtaining the results reported in Sections 3, 4, and 5. The second type of test-bed is a less abstract problem designed to simultaneously test several aspects of a learning algorithm in a problem that can be given an intuitively clear physical interpretation. This second type of test-bed helps ensure that the abstract text-beds deal with realistic issues.

The results reported in Section 2 were obtained using tasks that we interpret as types of spatial-learning problems. Although these "landmark-learning" problems did not result from an attempt to model real spatial learning, we found that these tasks provide an easily visualized way to study issues that are relevant to more difficult control problems (where the "physical" space becomes the abstract state space of a dynamical system). Our original motivation for studying landmark-learning problems was that the spatial setting allowed us to define problems that we knew could be solved by networks restricted to implementing linear mappings. A side benefit of these experiments, however, has been their suggestiveness about the evolutionary origin of learning and its relation to the taxes and trophisms exhibited by simple goal-seeking organisms. Other results (reported in Section 6) were obtained using a more complex test-bed that requires the control of a simulated physical dynamical system. In this case, the problem is to learn to balance a pole under conditions that created a difficult credit-assignment problem. In designing these test-beds, we attempted to focus upon issues that we believe are of central importance in the design of efficient and extensible learning algorithms.

Although this research was obviously influenced by an interest in the neural basis of learning, we did not attempt to model specific neural processes. The adaptive components we studied have many neuronlike attributes and were motivated by Klopf's neural hypothesis, but we purposefully refer to them as "adaptive elements" rather than neural models. We take this position because we rely upon the problem-solving capabilities of our systems to justify their study rather than upon neuroanatomical or neurophysiological data. However, we hope that our research is suggestive for neuroscientists who are more qualified than we are to determine how strongly these constructions can be related to neural systems. In particular, we think that 1) the role of the random component of neural discharge needs to be reexamined in light of its possible role in search processes* and 2) the possible existence of associative reinforcement learning at the single cell level, as hypothesized by Klopf, needs to be experimentally examined.

Reinforcement-Learning versus Error-Correction Learning

One of the most important distinctions relevant to the research reported here is the distinction between reinforcement learning and error-correction learning. This distinction can be most clearly made in terms of the quality of the information supplied to the learning system by its environment. This information may range from explicit specification of the actions that the system is required to perform to unreliable and infrequent assessments of certain distant consequences of the system's actions. In the first case, the learning system need only remember what it is told, whereas in the second case, the system must somehow discover what actions have consequences that lead to improving performance.

When there is a "teacher" in the environment that can tell the learning system exactly what action it should take for each input, then "learning" is easy. This amounts to the rote storage of information, something that conventional computer memory systems accomplish very efficiently. A less knowledgeable teacher may know the correct actions for just some of the input situations. Under these circumstances, a rote storage method that provides some form of generalization may permit correct extrapolation of the teacher's knowledge to a broader class of situations. This type of learning problem has been extensively studied as "supervised learning pattern classification" (see, for example, Duda and Hart, 1974). The teacher provides the learning system with a set of input patterns together with their correct classifications (e.g., a selection of exemplars and counterexemplars of each class),

* Hinton and Sejnowski (1983) make a similar suggestion.

and the learning system must correctly classify these samples while extending, via its generalization capability, the classification of the samples to the set of all possible input patterns. Algorithms such as that used by the Adaline (Adaptive linear element; Widrow and Hoff, 1960) or the Perceptron (Rosenblatt, 1962) are examples of these methods. They can be regarded as *error-correction* methods that adjust parameters so as to reduce the discrepancy between how they respond and how their teachers instruct them to respond. Artificial Intelligence researchers study higher-level versions of this same type of problem as "learning from examples," "concept formation," or "inductive inference" (Cohen and Feigenbaum, 1982). Although the capacity to generalize is important for both efficient learning and efficient information storage, we do not believe that all aspects of learning can be accounted for by mechanisms that require such explicit information, even if it is required for only a subset of the possible input situations.

More powerful learning capabilities result from the combination of information storage methods with some form of problem-solving or "discovery" process. The problem-solving process determines what information needs to be stored in order to efficiently solve a given problem. The role of the "teacher" is played by the system's own problem-solving experience. What is needed for implementing this problem-solving component is a strategy variously called "blind variation and selective survival" (Campbell, 1960), "trial-and-error search," or more recently by AI researchers, "generate and test." This type of process generates trials whose consequences are unforseeable at the time they are generated. These trials are then evaluated and selected according to their consequences in furthering a given problem's solution. Trials need only be "blind" in the sense of not being based upon *complete* knowledge of the outcome of a trial before it is generated. Any amount of knowledge, present initially or acquired during the problem-solving process, may be used to generate trials with high likelihood of improving problem-solving performance, but true discovery requires at least some initial doubt.

By a reinforcement-learning system we mean a system employing this type of generation and selection process. The information from the environment rewards or punishes actions that were made, but does not instruct the learning system as to what any correct action would have been. The teaching information from the environment is a scalar evaluation or reinforcement signal rather than the more extensive teaching information required in error-correction learning. Reinforcement-learning systems are therefore able to improve their performance in environments that provide information that is of lower quality than required by error-correction systems. This is not to say that error correction plays no part in reinforcement learning. A reinforcement-learning system must internally determine estimates of error signals based on some form of memory of its past actions and past performance. The point is that the learning system must do this for itself rather than rely on so helpful an environment.

We might also call reinforcement learning *selectional* and error-correction learning *instructional*. These terms have their origin in immunology where a selectional theory of antibody formation is one in which specific antibodies are selected from an extensive existing repertoire of potential antibodies by the action of antigens. Antibodies so selected then proliferate. An instructional theory, on the other hand, holds that the structure of an antigen directly alters a molecule to make it an effective antibody; that is, the antigen acts as a kind of conformational template. This use of the term selectional, and its use to describe neural theories of learning (e.g., Edelman, 1978), refers to selection from a repertoire of *simultaneously* existing candidates. However, the term selectional can equally describe processes in which the repertoire of candidates is generated *over time*. Selection from simultaneously existing candidates is undoubtedly an important mechanism of development and learning, but the research reported here focuses on selectional mechanisms in which variety is generated over time.

Additionally, our research has focussed on *associative* reinforcement learning in which the learning system receives *neutral input*, i.e., input that is not teaching or evaluative information, in addition to reinforcement input. The task is not just to discover an optimal action (as it is, for example, in studies of function optimization algorithms) but to discover an optimal mapping from input to output. The work of Klopf (1972, 1982) has been instrumental in making us aware of the crucial difference between associative reinforcement learning and both pure search and error-correction learning. As Klopf suggests, associative reinforcement learning has not received as much attention by cyberneticians and AI researchers as have pure search and error-correction learning. Yet properly understood, associative reinforcement learning is an obvious way to improve the performance of a problem-solving system. Its essence is the "caching" of search results in an associative memory so that future search is converted into simple memory access. The importance of this process is appreciated by AI researchers (see, for example, Lanat, Hayes-Roth, and Klahr, 1979), but its use in AI systems is not yet widespread. We believe that if this process pervasively occurs from the lowest levels of a problem-solving system up to the highest, and is combined with the well-established generalization capabilities and noise tolerance of associative networks, then the resulting systems will possess impressive ability to improve performance with experience.

The Credit-Assignment Problem

There are major problems, however, that need to be overcome if associative reinforcement learning is to be useful in complex learning tasks. Foremost among these is the *credit-assignment problem* (Minsky, 1961). The credit-assignment problem for reinforcement-learning systems is the problem of correctly assigning credit or blame to each of the actions and internal decisions that contributed to the overall evaluation received. This problem can become exceedingly difficult either as overall evaluations become more infrequent, making it less clear which overt actions were responsible for changes in performance, or as the learning system becomes more complex, making it less clear which internal decisions were responsible.

This suggests that it is useful to divide the credit-assignment problem for complex learning systems into two subproblems. One subproblem is that of converting the overall evaluation for a sequence of steps into an evaluation for each step. The other subproblem is that of using the evaluation of each step to assign credit to the internal processes of the learning system that determined the action selected on that step. One can call the first subproblem the ***temporal credit-assignment problem*** and the second subproblem the ***structural credit-assignment problem.*** The research reported here concerned both types of credit-assignment problems, although the major effort was devoted to the temporal credit-assignment problem.

The cause of either type of credit-assignment problem is initial uncertainty about the causal microstructure of the interacting system and environment. Unless we are willing to assume the existence of sufficient *a priori* knowledge either built into the system or into an external teacher (which we are *not* willing to assume), this uncertainty is unavoidable, and mechanisms must be devised that can reduce it over time. One approach that has been studied involves stochastic search algorithms that are capable of extracting specific types of statistical regularities from their interactions with random environments. These algorithms are related to those of the theory of stochastic learning automata (Narendra and Thathachar, 1974). The concentration upon *associative* reinforcement learning may be considered another prong in this attack on uncertainty. Neutral input can be used to divide a large problem into many subproblems in each of which the uncertainty is more manageable. A third aspect of our approach to uncertainty has been the development of an *adaptive heuristic critic*, in the form of a neuronlike element, that is capable of constructing from neutral environmental signals an evaluation function that is of higher quality than any that is available directly.

The adaptive critic is an extension of our previous studies of classical conditioning (Sutton and Barto, 1981; Barto and Sutton, 1982) and was influenced by Klopf's (1972) concept of "generalized reinforcement" at the level of single neuronlike elements. It turns out also to be closely related to a method used in the famous checkers playing program developed by Samuel (1959). If particular stimuli are regularly followed by high reinforcement, then a good critic should assign credit to behavior occurring near the time of occurrence of those stimuli, rather than to behavior occurring near the time of the high reinforcement. Credit should be assigned at the earlier time because that was when the critic was first informed that the high reinforcement was coming, and thus the most likely time of the action that caused it. Whereas *primary reinforcement* may be regarded as the credit delivered by externally supplied reinforcement, *secondary reinforcement* is credit delivered by stimuli that have acquired reinforcing properties through this kind of association with primary reinforcement. It is also possible for secondary reinforcers to be learned by association with previously established secondary reinforcers rather than only with primary reinforcers. Thus, secondary reinforcement can be "chained" backwards in time. Secondary reinforcement mechanisms can help overcome the weaknesses of the "recency heuristic" in reinforcement learning. Minsky (1961) mentioned the addition of special devices to reinforcement-learning machines for this purpose.

In Section 6 we describe a simulation experiment in which the adaptive critic coupled to an associative reinforcement mechanism was applied to a control task having a difficult temporal credit-assignment problem (a specific form of the pole-balancing problem). On test runs, this system's performance was dramatically superior to that of an algorithm previously proposed by others.

We believe that this type of adaptive critic will play a major role in solving structural credit-assignment problems as well as temporal credit-assignment problems. If every reinforcement-learning element incorporates its own local instance of an adaptive critic, then whatever signals happen to be available to the element can serve as the raw information from which the adaptive critic can construct a rich, local reinforcement signal. This may be necessary to assign credit efficiently to the microdecisions that lead to overall network output. We have not yet experimented with adaptive critic elements as components of complex networks.

Some Clarifications

The notion of goal-seeking components, our particular formulations of such elements, and our modular, issue-driven methodology have led to several types of misunderstanding and criticism that we wish to discuss. One type of misunderstanding has arisen because we have not discussed how we see the approach we are taking fitting into an elaborate overall model of intelligent behavior. Consequently, there are a number of subjects upon which we have not commented. This silence upon a subject has sometimes been interpreted as an indication of our belief that that subject is not important in an overall view of intelligence. This interpretation is mistaken. We have purposefully not commented upon a number of issues simply because they have not yet arisen at the level of the problems we have so far considered. For more complex problems, more elaborate learning systems will be required, and we prefer not to bring elaborate mechanisms to bear upon problems that can be solved by simpler mechanisms.

One example of this type of misunderstanding arises because our test-beds to date have not involved the use of any but the most simple types of *a priori* knowledge. This has made it appear to some that we are philosophically inclined toward a *tabula rasa* view of learning and intelligence. This is not the case. On the contrary, we think that as much knowledge as is initially available ought to be brought to bear on any task. Since we are studying learning, however, we are interested in problems that cannot be efficiently solved by relying *exclusively* upon existing knowledge. We intend the methods we are studying to be suitable for filling-in missing knowledge whether that residue is all of the relevant knowledge or only a small part of it.

Another cause of misunderstanding is the fact that in our studies to date we have shown a reinforcement signal being delivered to a learning system along a single channel that is distinct from the channels by which the system receives other information. This should be regarded as a convenient abstraction. We could have formalized the problem as one in which there is only one kind of channel, and the learning system internally determines the reinforcement value of the input patterns that arrive over these channels. According to this view, the learning system would have a *preference ordering* over the set of its possible stimuli, and internally-computed reinforcement values would code this preference ordering. Although taking this latter view would help dispel several types of criticism, the two possibilities are actually equivalent in all respects that are important at the present stage of our research.* The only difference between the two is the location of the boundary between the learning system and its environment. The signal carried by our formal reinforcement pathway represents a measure of performance that, in physical terms, could be delivered to the learning system in a variety of ways.

A related criticism is that we are ignoring motivational factors whereby an organism modifies the reinforcing quality of stimuli based upon its internal state. It is correct that we have not focused on motivational factors, but the possibility of handling these factors is not precluded by our formulation. Any number of mechanisms can be interposed between externally supplied primary reinforcement and the reinforcement that is effective at the

* We have not yet considered the case of partially ordered inputs to elements.

sites of learning. The adaptive critic is one such mechanism. At present, we assume that the effects of other mechanisms modulating reinforcement are already included in what we label the primary reinforcement signal. We have not addressed mechanisms that would correspond to motivational mechanisms in animals because we believe that problems that are much more fundamental need to be solved first.

A final criticism arises because we insist on endowing individual elements with learning abilities that others prefer to place in centralized mechanisms. If one wishes to interpret single adaptive elements as neurons, then our view seems to deny the existence of any structure that would correspond to the centralized reinforcement areas of animal brains. Again, this criticism results from too literal a reading of our formulations. The environment of an adaptive element includes the rest of the network as well as the external environment. This network can implement mechanisms for generating element-level reinforcement. These mechanisms may be centralized, distributed, or both.

Related Theoretical Fields

The research reported here is related to several theoretical fields. We indicated in an early article (Barto, Sutton, and Brouwer, 1981) that the algorithms implemented by our adaptive elements combine aspects of stochastic approximation algorithms and stochastic learning automaton algorithms. The precise relationship between these types of algorithms appears not to be well understood mathematically, and we have not seen analyses within either of these areas of the type of synthesis that we have been studying. Our research indicates that the theorems appropriate for this synthesis are not simple consequences of the theory in either area. However, our research did not originate within either of these theoretical traditions, and there is much that we have to learn about them. This report indicates the relationship between our work and these fields more carefully than did earlier work. We hope that our results suggest novel directions for future research by speicalists in these fields.

Another related theoretical area is Markovian Decision Theory (e.g., Derman, 1970; Mine and Osaki, 1970). Our justification of the adaptive critic algorithm involves concepts from this area, and we intend to explore these connections more fully in the future. At present, we do not know of algorithms similar to the adaptive critic algorithm other than those of Samuel (1959) and Witten (1977), which are cited in the appropriate places in Section 6.

Overview of the Report

Section 2 presents results of our experiments with layered networks applied to nonlinear learning problems. These networks adaptively create new features by combining existing features. Also included is a brief review of past layered-network studies that serves to place our own approach in perspective. Section 3 describes experiments with nonassociative reinforcement learning. By excluding the associative aspect, we are able to focus on the basic search problems and to compare a number of algorithms, including several well-studied stochastic learning automaton algorithms. We propose a new type of algorithm and present results that strongely suggest its superior learning rate. In Section 4 we extend the algorithms discussed in Section 3 to associative reinforcement learning tasks. We describe experiments that compare a number of algorithms on a number of tasks. These results show that some apparently intuitively satisfying algorithms have serious difficulties compared to other algorithms. We describe some new algorithms that overcome these problems. The experiments of Section 5 deal with the effects on learning of delayed reinforcement. These results generally support the expected consequences of delayed reinforcement but also reveal specific limitations of certain types of algorithms. Section 6 provides justification for the design of an adaptive heuristic critic element for implementing secondary reinforcement and illustrates its utility in a learning task requiring the system to learn to balance a pole. Section 7 contains concluding remarks.

Section 2
LAYERED NETWORKS
AND THE GENERATION OF NEW FEATURES

Introduction

When they were first studied, adaptive networks seemed to offer the promise of providing a means for adaptively creating novel features with which to represent a given problem. Even if each adaptive element could only adjust the weightings of its existing input channels, other adaptive elements supplying the signals on these channels could conceivably redefine the meaning of these signals in a manner appropriate for the problem at hand. Unfortunately, despite considerable effort, this promise of adaptive networks has not yet been convincingly demonstrated. Another way of stating the problem is that although networks of linear threshold elements can be constructed to implement *any* input/output function, there has never been a satisfactory solution to the problem of having a network efficiently *learn* to implement desired nonlinear functions whose implementation details are initially unknown. In particular, learning algorithms that work for single layers of adaptive elements cannot be easily extended to multilayer networks. Yet if one wishes to use adaptive networks to provide problem-solving systems with open-ended representational capabilities, the problem of obtaining learning in complicated networks must be solved. We therefore view the problem of learning in multilayer networks to be central to the problem of establishing the utility of adaptive networks.

In this section we describe some results of our experiments with layered networks of goal-seeking components. These results provided initial indications that networks consisting of these types of components are able to learn to implement desired nonlinear functions by creating useful new features. Although further research is needed to 1) optimize element design for use in layered networks, and 2) investigate this behavior for larger, more complex problems, these initial results indicate that the approach we are taking may have the potential for circumventing this classical problem with adaptive networks.

Briefly, we view the difficulty in obtaining learning by multilayered networks to be chiefly due to the difficult structural credit-assignment problem (see Section 1) that exists for complex networks. Somehow, evaluations of the behavior of the entire network must be apportioned correctly to the individual elements, and to the individual weights, that were responsible for that behavior. Rather than designing a centralized agent that knows enough about the network to correctly apportion credit, our approach is to endow each adaptive element with learning capabilities that are sophisticated enough to enable it to increase *its own performance* in the face of considerable uncertainty by using any information that is locally available. Uncertainty is present since the influence of any single element on *network* evaluation will generally not be deterministic from that element's point of view. With a few exceptions, adaptive elements studied in the past were not designed for this type of task.* We think that the ability of network components to perform well in associative renforcement learning tasks under uncertainty is a prerequisite for obtaining nontrivial learning by complex networks.

There is an extensive history of attempts to extend single-layer learning results to networks having multiple layers. We first briefly review these studies in order to extract general principles and to place the approach that uses goal-seeking elements in its proper context. We then review the results we have obtained for layered networks.

Review of Layered Network Studies

Assume that a multilayered network has been designed by defining the output of each element as a parameterized function of its input, and by specifying all of the interconnections among the elements. The question that is associated with learning in such a system is: What values should be assigned to the parameters of the network in order to implement some desired input/output function? Numerous approaches have been taken to this problem.

* One of the results reported in Section 4 is that those few adaptive elements that were designed for this type of task are not very good at solving it.

Direct Search — The most straightforward approach is to directly search the space of the network's parameters for those values that maximize some measure of network performance (e.g., a measure that is maximized when the classification errors of the network's output elements are minimized). Gilstrap (1971), who has pursued this direct-search approach, used guided random search methods that can be effective under conditions encountered with multilayered networks. These conditions are high dimensionality of the parameter space, multimodality of the evaluation function (i.e., the existence of "false peaks"), and noisy input information.

Whatever the search method used, however, directly searching a space of dimension equal to the number of parameters in an entire network is an extremely time-consuming process for all but the simplest cases. This direct-search strategy is not suited to real-time learning, not only because it is slow, but also because it requires a centralized control strategy and off-line evaluation of network performance. The view of the problem adopted in this approach does, however, provide a formal view of what needs to be achieved (or approximated) even if it does not lead directly to practical, real-time algorithms.

Single Adaptive Layer — Results from early adaptive network research concern layered networks in which the elements of only one layer can adapt. An example of such a network is the Perceptron of Rosenblatt (1962). The Perceptron is a two-layer network (three-layer in Rosenblatt's terms) in which the elements of the first layer are connected randomly to the lowest-level pattern features (the binary pixels of a visual pattern). The idea is that if enough elements were so connected, a representation would be implemented that would permit the second layer to linearly form the required discrimination via an error-correction learning procedure. Another example of a layered network having a single adaptive layer employing an error-correction algorithm is the network of Adalines defined by Widrow (1962).

There are obvious difficulties in extending these error-correction techniques to layered networks. Let us distinguish a network's *output elements* from its *interior elements*. Output elements are those whose activity is directly visible to the network's environment and that are required to assume certain values for various input patterns provided to the network. Interior elements are those whose activity is not directly visible and that are somehow to provide an encoding of input signals that will allow the output elements to respond correctly. The Achilles' heel of pure error-correction elements as network components is that they can only learn if supplied with individual desired responses or error signals. Although in many tasks it may be possible for the network's "teacher" to provide error signals to the network's output elements (since it is these that define the network's visible response), it may not be possible for this teacher to provide analogous error signals to the interior elements without *a priori* knowledge of the implementation details of the desired input/output function. If this knowledge were available, then the problem would be quite different from the ones in which we are interested: It would be a programming problem, not a learning problem.*

A variety of methods have been proposed in which error signals for interior elements are inferred from error signals available for the output elements. One can view some of the methods described below in this way.

Generating New Elements — Some methods rely on the generation of new elements rather than on the adjustment of the parameters of existing elements. Methods that generate new elements generally divide the learning process into two stages. In the first stage, the parameters (weights) of the first layer (i.e., the layer that directly receives the external input signals) are held constant while one of the familiar single-layer learning algorithms is used by the second layer. In the second stage, the second layer is held constant while new elements are added to the first layer. Those elements whose outputs are not significantly influencing the second layer might be discarded to limit the number of elements.

Selfridge's Pandemonium provides an example of this two-stage learning process (Sel-

* By this remark we do not mean to deride efforts to program network architectures to produce real-time solutions to difficult computational problems. Much of the current resurgence of interest in the network approach stems from these demonstrations of network utility; for example, Feldman (1981) and Ballard's (1981) work on networks for low-level, real-time vision; Hinton's (1981) work on knowledge representation; and Rumelhart and McClelland's word recognition network (McClelland and Rumelhart, 1981; Rumelhart and McClelland, 1982).

fridge, 1959). The second stage begins with the discarding of Layer-1 elements of low worth, which Selfridge defined as the sum of an element's output weights, i.e., the weights through which the element's activity reaches other elements. Two methods are used for replacing low-worth elements, called "mutated fission" and "conjugation." Mutated fission consists of randomly selecting one of the remaining Layer-1 elements and altering some of its parameters, also at random. To form a new element by conjugation, two of the remaining elements are randomly chosen and logically combined by selecting one of ten nontrivial logical functions. An element is added that computes this logical function of the output of the two chosen Layer-1 elements. The classifier system of Holland (1980) is probably the most highly developed example of generating new elements via this type of "genetic recombination" process.

Klopf and Gose (1969) compared three methods for measuring the worth of the elements in the first layer of a two-layered pattern recognition network. One method is to use the size of an element's output weight as a measure similar to Selfridge's worth function. The second method is to use the product of the output weight and the element's output value. A third method is to use the absolute value of the cross-correlation between the output of the Layer-1 element and the desired network output. Their results show that for the tasks they considered the product of the output weight and the output produces better performance than the output weight alone, and both are better than the cross-correlation measure.

In addition to these examples, many other systems employing similar principles have been applied to the problem of feature selection in the pattern recognition domain. In some cases, a repertoire of possible features is set up initially, and new features are chosen from this set (e.g., Samuel, 1959). Often a method of feature ranking is employed to determine the relative usefulness of features for a particular problem.

Beam Search — Methods for generating new elements are attempts to avoid the combinatorial explosion that would result from having an element for each of the possible combinations of available signals. The heuristic employed is that useful higher-order features will tend to be compositions of useful lower-order features. A technique using this heuristic may be viewed as a type of *beam search* (see Barr and Feigenbaum, 1981). The search is conducted by forming all pairwise (for example) combinations of lower-order features at each stage, and then removing from consideration all but a certain number of them before forming the next stage's combinations. At each stage, the number of features remaining is the *beam width* of the search. A beam search is not guaranteed to result in an optimal solution but can be efficient if the beam is sufficiently narrow.

Although not usually associated with networks of adaptive elements, beam search can obviously be related to layered networks. Ivakhnenko's (1971) Group Method of Data Handling, for example, is a method for constructing a layered network using what is essentially a beam search. He assumes that a set of desired situation-response pairs is available *a priori*. This set is divided into a training set and a testing set. The algorithm proceeds by developing one layer at a time. For the first layer, the original input components are divided into disjoint pairs, each of which is represented by an element that computes a quadratic polynomial of that pair of components. A regression technique is used to determine the coefficients of each polynomial for which the polynomial best approximates the desired mapping defined by the situation-response pairs in the training set. The testing set is then used to calculate the mean-square error for each element, and elements with an error above a specific threshold are discarded. Now the outputs of the remaining elements become the input components to the next layer, and the procedure is repeated to determine the polynomials and their coefficients for the elements in the second layer. The process is repeated for successive layers.

Since Ivakhnenko's method employs a centralized control structure both to select elements and to perform the regression analysis, it is not an adaptive network of the sort we are studying. Nevertheless, the principle it illustrates (i.e., the idea of beam search) is very well suited to completely parallel implementation by networks of adaptive elements.

Open-loop Adjustment — Another approach is to train the first layer of a two-layered network in isolation, independently of the second layer and of the network's performance on the required task. Such open-loop procedures are referred to as *clustering* methods. Clustering methods are based on certain assumptions, of which the most common can be expressed for layered networks as the assumption that the input patterns experienced by the network tend to be similar, according to some prespecified measure, to those in one of several sets of input patterns, or clusters. The dimensionality of a problem can be reduced if the elements of the first layer specify which cluster the current input pattern belongs to and present only the cluster label to the second layer rather than the original representation of the input pattern. This assumes that the same response is desired for all patterns that are members of one cluster.

Block, Nilsson, and Duda (1964) used a clustering algorithm to train the first layer of a two-layered network. They suggested that their method should be used to fully train the first layer before the second layer was trained. Fukushima's Cognitron (1973) and Neocognitron (1980) are other examples of clustering algorithms implemented as networks. In these networks, the elements within a single layer are arranged in an array, and an element is selected for adjustment if its output is sufficiently in excess of the outputs of the neighboring elements. Several such elements are selected from each layer. The weight associated with each element input pathway is adjusted so as to place its value closer to the current value of the input signal on that pathway. This causes the weight vector to more closely match the current input pattern.

In these examples, and others, one can see a common problem and similar mechanisms for dealing with it. If the network elements use the same learning algorithm and are presented with the same patterns, then they will tend to respond to, and thus to represent, the situations of the same cluster. To force the elements to tune to a variety of clusters, some form of additional communication between the elements must be included. This communication is usually used to suppress the learning mechanisms in untuned elements for any situation that is already represented by a previously tuned element. In more "neural" terms, this can be described as a type of lateral inhibition, and there are many models that use this type of mechanism. We refer to the general problem as one of *enforcement of variety.*

Adjustment Based on Performance — Some open-loop methods, e.g., that of Block et al. (1964) mentioned above, only stop once a set of clusters has been found from which all of the experienced input patterns can be reconstituted. This implies that possibly very many clusters, and thus elements, are required. Certainly, for many problems such a large set is not necessary, specifically for problems whose solutions do not associate a different response with each input. Only those features that discriminate between inputs requiring different responses need to be developed by the elements in the initial layers. In other words, the need is not just to form clusters of input patterns but to form clusters that are *useful* in terms of the network's interaction with its environment. In order to accomplish this, the initial layers must use the information contained in an error or evaluation signal that is provided by the network's environment. The problem, as discussed above, is that this error or evaluation is directly a function only of the network's output elements. Somehow this information must be used to tune the interior elements.

The simplest way of doing this is to adjust a randomly chosen element when an error is made by an output element. This approach was analyzed by Alder (1975) who proved an extension of the Perceptron convergence theorem for layered networks. As he pointed out, however, the algorithm was "less than efficient."

Another method for selecting elements to adjust is to select those elements that would require the least amount of change to correct the network's error. Widrow used this method in networks consisting of two layers of Adalines (1962). Two notable characteristics of Widrow's method of training two-layered networks are: (1) the first layer learns faster than the second, and (2) responsibility for error is assigned to those elements that can most easily correct it. This algorithm and similar ones (e.g., Stafford, 1963) require a rather sophisticated agent to conduct the training of the interior elements.

In some cases, the sophistication required by this agent can be reduced if the network structure is sufficiently constrained. Widrow's (1962) study of Madalines (Multiple Adalines) can be interpreted in this way. A Madaline is a single layer of Adalines and an additional element in a second layer that computes a fixed logical function of the output signals produced by the Adalines. The output of this element is the network's response.

Typical logical functions are majority-rule and logical OR. The method of training the Madaline depends on the logical function that is used. For example, the following procedure is used if the second layer consists of a majority-rule element. An input pattern is presented and the Madaline's response is compared to the desired response. If they are equal, no adaptation occurs. If the desired response is $+1$ and a majority of the Adalines generated -1, then enough of those Adalines are adjusted to change the majority decision from -1 to $+1$. The Adalines selected for adjustment are those that made the wrong response and whose weighted sums, before thresholding, were closest to zero. For a desired response of -1 the same procedure is followed using the opposite signs. Widrow described this process as a way of assigning responsibility to those elements that could "most easily assume it." Methods that appear related to this have been discussed by Reilly, Cooper, and Elbaum (1982) and Hampson and Kibler (1982) and seem to offer promising ways of using networks for nonlinear pattern classification.

Networks of Reinforcement Learning Elements — Some of the aforementioned methods represent attempts to extend error-correction methods to all elements of the network, for example, by restricting network operation so that desired responses for interior elements might be deduced. Another approach is to use elements that do not require desired responses or error signals but that implement reinforcement-learning algorithms. Such elements are capable of improving performance with respect to an evaluation signal that assesses the collective activity of all of the network components. Since the activity of any individual component does not have a deterministic influence over the global evaluation signal, each element must be able to learn under a high degree of uncertainty, and the entire process requires "statistical cooperation" (Farley and Clark, 1954) among the network elements.

Our own approach and that of Klopf (1972, 1982) fall into this category, and we know of only a few earlier studies that are similar. In his Ph.D. thesis, Minsky (1954) described the SNARC (Stochastic Neural-Analog Reinforcement Calculator) which he constructed in 1951. It was a network of adaptive components that roughly corresponded to synapses rather than to entire neurons. These components transmitted incoming pulses according to a probability that was altered by subsequent global evaluation. If a pulse was transmitted and reward followed, then the transmission probability was increased, etc. In modern terminology, each "synapse" was implemented as a stochastic learning automaton (Narendra and Thathachar, 1974). Minsky applied his network to a maze-learning task and noted that it "displayed gratifying ability." Farley and Clark (1954) experimented with adaptive elements that are very similar to our adaptive elements. They have a set of adjustable weights, emit activity generated by a noisy threshold, and use a similar weight-update rule. In simulation experiments, networks of these elements were able to solve some simple discrimination tasks.

Both of these studies predated any use of error-correction learning rules in network studies. Consequently, Minsky, Farley, and Clark were not attempting to solve the layered-network problem since it had not yet surfaced. Their networks were layered (in fact, they were recurrently connected), and they did appear to learn effectively (although no systematic exploration of these abilities was undertaken). This approach using statistical cooperation seems to have been quickly overshadowed by an interest in the generalization capabilities of these networks — an interest that contributed to the development of the field of pattern classification. Farley and Clark (1954), for example, paid no attention to the fact that the interior elements of their network were capable of learning but instead focussed on generalization ideas which by now have become routine.

Minsky, on the other hand, seems to have been impressed with the potential difficulty of the credit-assignment problem inherent in this approach. As tasks become larger and more difficult, the unstructured statistical cooperation approach becomes inadequate. Heuristics more powerful than "recency" and "frequency" need to be employed to assign credit or blame to individual components of a decision-making process. He proposed the development of secondary reinforcement mechanisms and the use of some rich, local form of reinforcement (Minsky, 1961). He was aware, however, of the relevance of reinforcement learning for layered networks. Minsky and Papert (1969), for example, remarked that

> It ought to be possible to devise a training algorithm to optimize the weights of this [layered network] using, say, the magnitude of a reinforcement to communicate to the net the cost of an error. We have not investigated this. (p. 206)

We continue to find Minsky's early remarks on the problems of adaptive networks to be very penetrating, and they have obviously influenced our own approach.

Widrow, too, became aware of the relevance of reinforcement learning for layered networks. Widrow, Narendra, and Maitra (1973) presented an extension of the Adaline algorithm to allow it to do a form of reinforcement learning which they called "bootstrap adaptation." They remarked that this extension may permit the elements to learn as components of layered networks, but we know of no attempts to demonstrate this.

The approach proposed by Klopf (1972, 1982), in which components are seen as self-interested agents, also falls into this category. By formulating the goal of these self-interested agents as the maximization of certain variables (a condition Klopf calls "heterostasis" to distinguish it from homeostasis, which refers to an equilibrium condition), Klopf identified one of the key distinctions between supervised learning and reinforcement learning. Klopf also clearly pointed out the importance of this type of learning for layered networks.

Statistical Mechanical Methods — Before concluding this review of layered-network studies, it is appropriate to mention recent research that is relevant to the problem of learning in layered networks. Several researchers are currently studying networks of neuronlike elements that operate probabilistically and can be analyzed by methods analogous to those of statistical mechanics (Hopfield, 1982; Hinton and Sejnowski, 1983; D. Geman and S. Geman, 1983; S. Geman, 1984). It is not yet clear exactly what relationship exists between these methods and those discussed in the previous subsection requiring statistical cooperation. It is also too early to evaluate the utility of these methods (they may be too slow to yield practical real-time algorithms). They are very intriguing, however, and may help provide a much needed theoretical basis for the study of adaptive networks.

Layered Networks of Goal-Seeking Components

From the overview of approaches to learning by layered networks provided above, it is clear that the approach we have studied might be regarded as a continuation of the statistical-cooperative approach that was pursued very briefly by earlier researchers. We believe the sparsity of these studies is not due to any special difficulties of this approach as compared to error-correction approaches, but rather to historical circumstances and to the general misunderstanding that existed (and continues to exist) about the distinction between reinforcement learning and error-correction learning (see the discussion in Section 1).

Using the idea of self-interested, goal-seeking components developed by Klopf (1972, 1982), we are able to view the problem of learning in layered networks in terms of the notion of cooperativity from the theory of games. Consider, for example, a two-layer case in which elements of Layer 1 receive input from the external environment but are not able to directly influence that environment. These elements can only influence Layer-2 elements which, in turn, are able to influence the environment but do not receive neutral (i.e., nonreinforcing) input directly from the environment. All elements receive the same reinforcement signal. To attain higher levels of reinforcement, Layer-1 elements "need" Layer-2 elements because without them Layer-1 elements have no influence over the reinforcement signal. For their part, Layer-2 elements need Layer-1 elements because without them, Layer-2 elements have no clue as to the state of the environment. Without this information, the relationship between the actions of Layer-2 elements and reinforcement will appear to have few regularities. If Layer-1 and Layer-2 elements can correctly coordinate their behavior, they each can enhance their own reinforcement levels. This is a form of cooperation in the game-theoretic sense.

Cooperativity can be achieved by statistically linking the activity of Layer-1 and Layer-2 elements. For example, a positive weight from Layer-1 element A to Layer-2 element B would increase the relative frequency of patterns of activity in which both A and B were active. We could say that A and B had entered into an agreement or had formed a partnership. The weight-update rules we have been studying are designed to alter the weights so that such agreements would be mutually beneficial to the elements involved. So linked, the elements A and B would form a *coalition*. Of course, since layered networks lack internal feedback pathways, we would see the formation of only restricted types of coalitions.

We now describe several computational experiments that we performed using layered networks. Among other things, these experiments illustrate this type of cooperativity. Additional details of these experiments, and descriptions of additional experiments, can be found in Barto, Anderson, and Sutton (1982) and Anderson (1982). After describing these experiments, we discuss their significance, some problems encountered, and their consequences for future research.

Description of the Tasks

The solution to the type of problem we considered requires a learning system to produce the "best" action for every input from its environment. This general formulation is shown in Figure 1. The current state of the environment determines the input to the system, which then applies its situation-to-action mapping (its control surface) to generate an action. The action then affects a change in the environmental state, which causes a change in network input. The critic evaluates the new input by computing a performance index, payoff, or reinforcement, based on the input or how the input is changing. The reinforcement is used to alter the system's mapping. After an appropriate mapping is learned, the reinforcement input is no longer required. The situation-to-action mapping is divided into an adaptive feature encoder and a feature-to-action mapping as in Figure 2. Note that the same reinforcement goes to both the adaptive feature encoder and the adaptive feature-to-action mapping.

This type of learning problem is greatly simplified compared to more general problems by the availability of a reinforcement signal at every time step that always evaluates the action made at the preceding step. In more general problems, an evaluation may be received only after the learning system has performed a long series of actions, thus introducing a difficult temporal credit-assignment problem. We temporarily adopted this simplification to allow us to focus on basic feature-generation issues. In Sections 5 and 6 we describe our

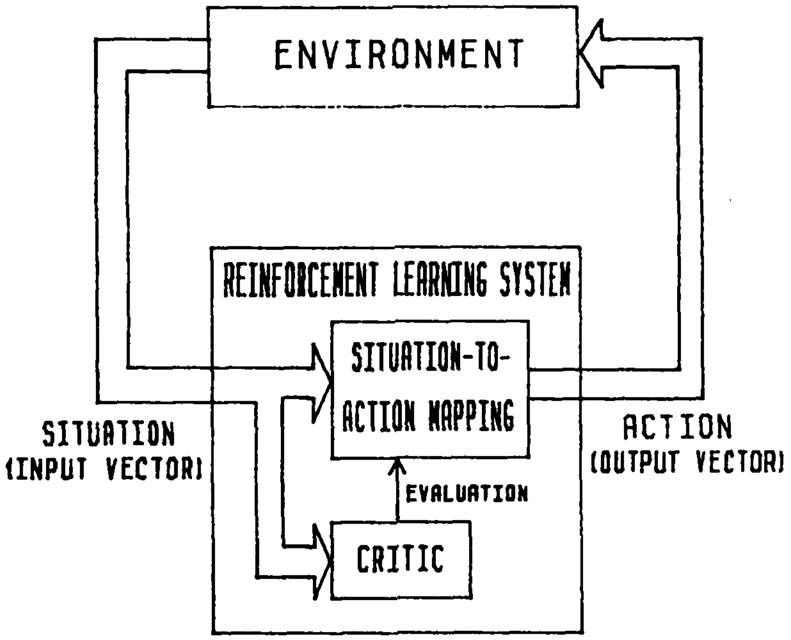


Figure 1

General View of a Reinforcement-Learning System.

research on the credit-assignment problem created by delayed evaluation of actions.

We have found that an interpretation of the above type of problem as a type of spatial learning problem (which we termed a "landmark-learning problem" in Barto and Sutton, 1981a) provides an easily visualized setting for studying some of the concepts that will be useful in more difficult tasks. In particular, one can regard the "space" in question as the state space of a dynamical system.

Nonlinear Landmark-Learning Problems

We briefly describe the simple linear landmark-learning problem and the single-layer network capable of solving it that was presented by Barto and Sutton (1981a).* We then describe an extension of this problem to one requiring a two-layer network for its

* Our presentation here differs slightly from that of Sutton and Barto (1981a). The symbols for the landmarks and the ordering of sensory input pathways to the network are different.

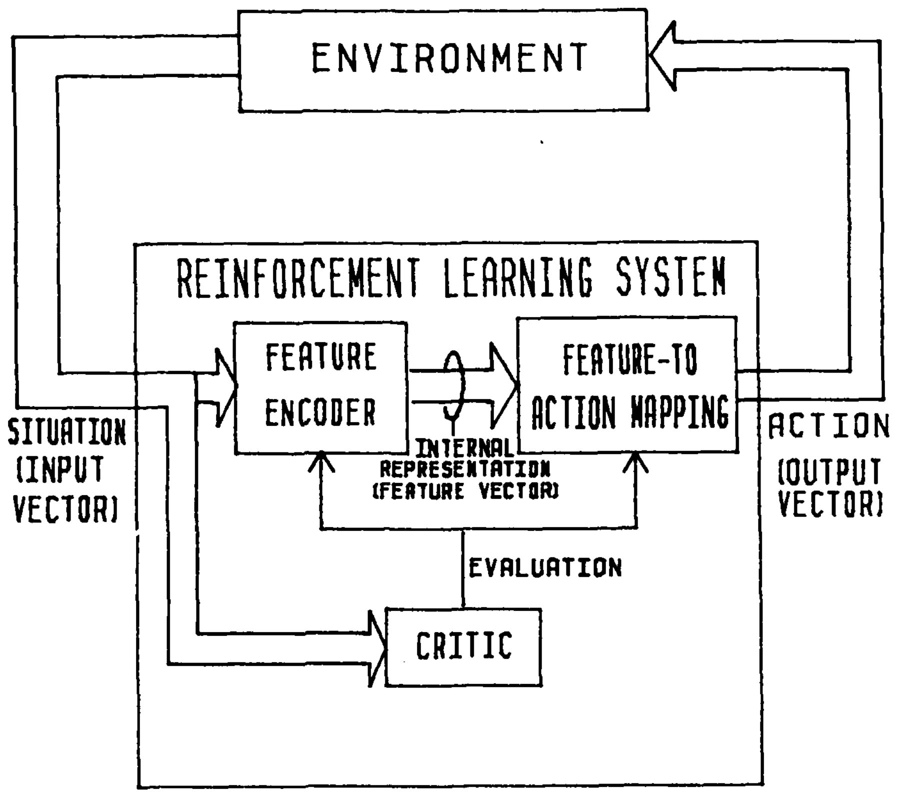

ENVIRONMENT
REINFORCEMENT LEARNING SYSTEM
FEATURE ENCODER
FEATURE-TO ACTION MAPPING
INTERNAL REPRESENTATION (FEATURE VECTOR)
SITUATION (INPUT VECTOR)
ACTION (OUTPUT VECTOR)
EVALUATION
CRITIC

Figure 2

Addition of Adaptive Feature-Encoding Mechanism.

solution. Figure 3A shows the environment of a simple "organism" which is represented by the asterisk. The tree in the center of the figure is the organism's target and emits an "attractant odor" (the payoff or reinforcement) whose strength decays with distance from the tree. Each of the landmarks at the cardinal points also emits a distinct "odor," decaying with distance, that does not act as an attractant (i.e., is neutral) but can serve as a cue to location in space. The organism's task in this environment is to approach the tree as efficiently as possible and remain in its vicinity. In order to do this, it acquires a control surface that tells the organism which way to go from every place in its environment. The inputs to this controller are the patterns of "odors" from the neutral landmarks and the central tree, and actions determine movement in space. Once in possession of an adequate control surface, the organism can use it to move directly to the place where the attractant

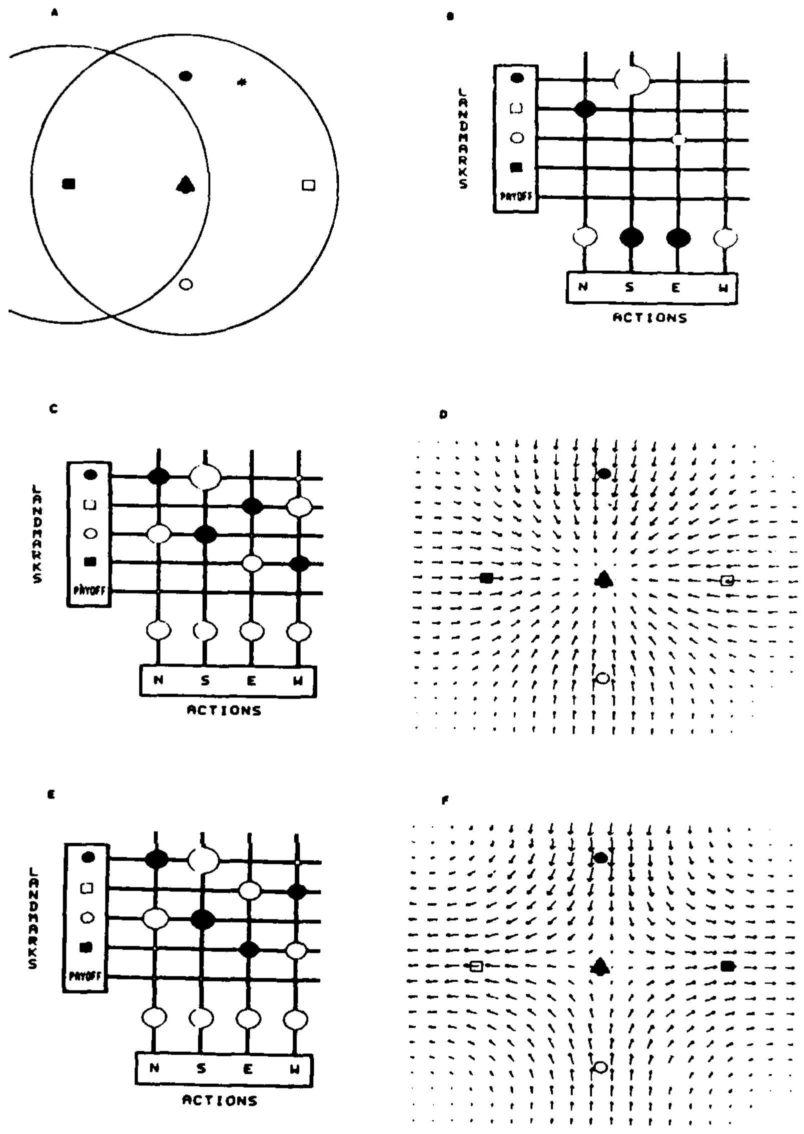


Figure 3

Linear Landmark-Learning Problem.

peak usually appeared even in the complete absence of the attractant distribution.

The organism's "nervous system" is the four-element network shown in Figure 3B. The four adaptive elements control motions in the respective cardinal directions. The control surface is stored as a matrix of weights connecting the neutral landmark inputs with the action-generating elements. The weights between input and output elements are shown as

circles centered on the intersections of the input pathways with the element "dendrites." Positive weights appear as hollow circles, and negative weights appear as shaded circles. The size of a circle codes the weight's magnitude. The action commanded by the network is to move north if Element 1 fires, south if Element 2 fires, etc. In case two nonopposing elements fire simultaneously, the appropriate compound move is made, e.g., northwest. Each move is a fixed distance and is always completed in one time step.

The weight associated with an input pathway from a given landmark to an element controlling movement in a particular direction increases if a step in that direction is taken in the presence of that landmark's signal *and* the resulting movement is up the attractant gradient. With sufficient experience, the organism learns respond to the "olfactory" cues at each place with the action that is optimal for that place. The operation of this network is fully described by Barto and Sutton (1981a). Figure 3C shows the network after it formed appropriate weights, and Figure 3D shows the results of learning as a vector field giving the expected direction of the network's first step for each position in space. This vector field is determined from the network's weight values and is never literally present in the environment.

In another experiment described by Barto and Sutton (1981a), the "organism" was allowed to learn in the environment just described, and then the box shaped landmarks were interchanged. The "organism" was initially misled by its sensory information but quickly relearned to the altered environment. Figure 3E gives the vector field showing how the "organism" was misled, and Figure 3F shows the network after it had relearned to the exchanged landmarks. When we changed the environment back to its original configuration, the network changed its associative matrix back to the original settings. Thus, it is clear that the network as described is not capable of maintaining both control surfaces at the same time. As it learns in an environment with a different configuration of the same landmarks, it "rewrites" its memory, erasing traces of previous learning. This suggests the following task, which turns out to be nonlinear in terms of the landmark signals.

The Two-Environment Landmark-Learning Problem — Figure 4A shows an environment containing two areas labeled "Region A" and "Region B." Corresponding landmarks produce the same sensory signals in both regions (e.g., the shaded box "smells" the same in both regions), but sensing a box landmark should produce movement in opposite directions in the two regions. That is, "hollow box in Region A" should be associated with movement east, but "hollow box in Region B" should be associated with movement west. Similarly, the correct associations for the shaded box depend on the region in which it is sensed. We considered the case in which there exist features, detectable by the network, that distinguish Region A from Region B. In the most general case, these distinguishing features may be complex patterns or relationships between more basic features, but for simplicity, and without undue loss of generality given our purposes, we simply assume that there is a sensor that is activated whenever the system is in Region A and one that is activated whenever it is in Region B. A signal from one of the region sensors must be capable of switching the effects of the two box landmarks on the east and west output elements in opposite senses. This cannot be accomplished by the sort of linear mapping the network shown in Figure 3B is capable of forming (this is proved in Barto, Anderson, and Sutton, 1982; the problem requires the computation of an analogue of the logical exclusive-or operation).

A signal is needed to distinguish the sensing of a landmark in Region A from the sensing of that same landmark in Region B, but we wished a network to form this signal on the basis of its experience rather than being provided with it from the start. Figure 4B shows a network consisting of two layers of adaptive elements. The output layer (Layer 2) shown at the bottom is identical to that shown in Figure 3B except that it has eight rather than four input pathways in addition to the reinforcement pathway. The input layer (Layer 1) consists of eight adaptive elements each receiving input from the four landmarks, the Region A and Region B indicators, and the reinforcement signal.

The eight elements of Layer 1 are organized in pairs: Elements 1 and 2, Elements 3 and 4, etc. The elements in each pair inhibit one another so that only the most strongly stimulated element of each pair can be active at any time. The large positive connection weights in Layer 1 are all set permanently to the same value. Consequently, before any

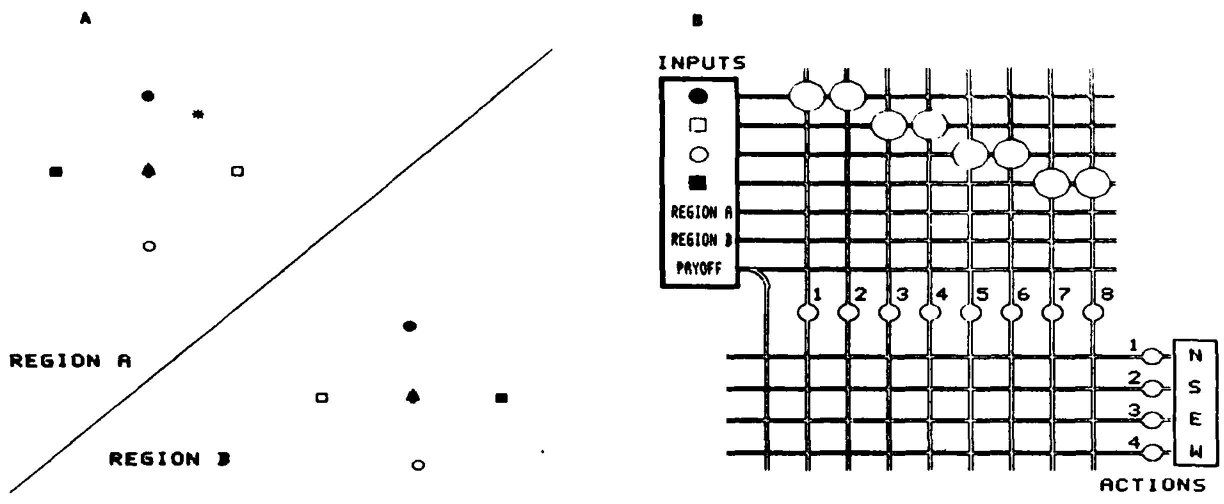


Figure 4

Two-Environment Landmark-Learning Problem.

learning takes place in Layer 1, the Layer-1 elements simply transmit the Layer-1 input signals to Layer 2, sometimes via one element of each pair and sometimes via the other (so that this network can also solve the linear problem described above). If the task cannot be solved linearly, then the paired elements differentiate, or "split," in terms of the input patterns to which they are tuned and the influences they exert on Layer 2. The Layer-2 elements are also paired so that at each time step only one element in each of the north/south and east/west pairs is active.

We first placed the network in Region A where it climbed the attractant distribution due to the presence of the tree and produced the trail shown in Figure 5A. At the same time, it formed associations between its stimulus patterns and the optimal actions. These associations are shown as a vector field in Figure 5B. Notice that the associations are correct for Region A but are incorrect for Region B. This is because the network's generalization from its experience in Region A to Region B is inappropriate due to the reversed box landmarks. The network that resulted from this experience is shown in Figure 5C. There was a tendency for one element of each of the Layer-1 element pairs to become tuned to respond strongly to various patterns of landmark "odors" in Region A and less strongly to these patterns outside of Region A (since positive weights form from the Region A sensor to these elements). These elements initially happened to be active more frequently and, as they began to be excited by the Region A input pathway, their probabilities of activity steadily increased. During this period, the connections from these elements to Layer 2 were established in a manner appropriate for moving in Region A. Thus a control surface appropriate for directing action in Region A was formed.

When the organism was then placed in Region B, the Region A control surface was accessed initially since the Layer-1 elements that were tuned to respond strongly to certain "odor" patterns in Region A also responded to these patterns in Region B, although less strongly. This resulted in the trail shown in Figure 5D and can be seen as the network's attempt to generalize its Region A experience to Region B. Having been placed north of the shaded circular landmark, the network proceeded almost directly south and west as a result of being correctly directed by the shaded circle and incorrectly directed by the hollow box. These actions were punished since the network moved down the attractant gradient in Region B. This tended to "erase" the Region A control surface. However, this also caused inhibitory connections to form from the Region B sensor to the elements selected in Region A. This steadily decreased the probability with which the elements selected in Region A became active in Region B. Then, whenever activation switched to the untuned element of a pair (and the probability of this steadily increased) *and* the network happened to move in the correct direction, then this element began to be tuned to respond to an "odor" pattern in Region B and therefore began to provide a signal to Layer 2 that could be associated with the correct actions for Region B. Consequently, the erasure of the Region A control information eventually stopped as new associations were formed appropriate for Region B (Figure 5F). Continued exploration resulted in the formation of the associations shown in Figure 5E as a vector field. New experience in Region A quickly reinstated any lost information (Figures 5G, 5H, 5I).

By examining Figure 5I, one can see that the Layer-1 elements tuned themselves to represent the environmental features as follows:

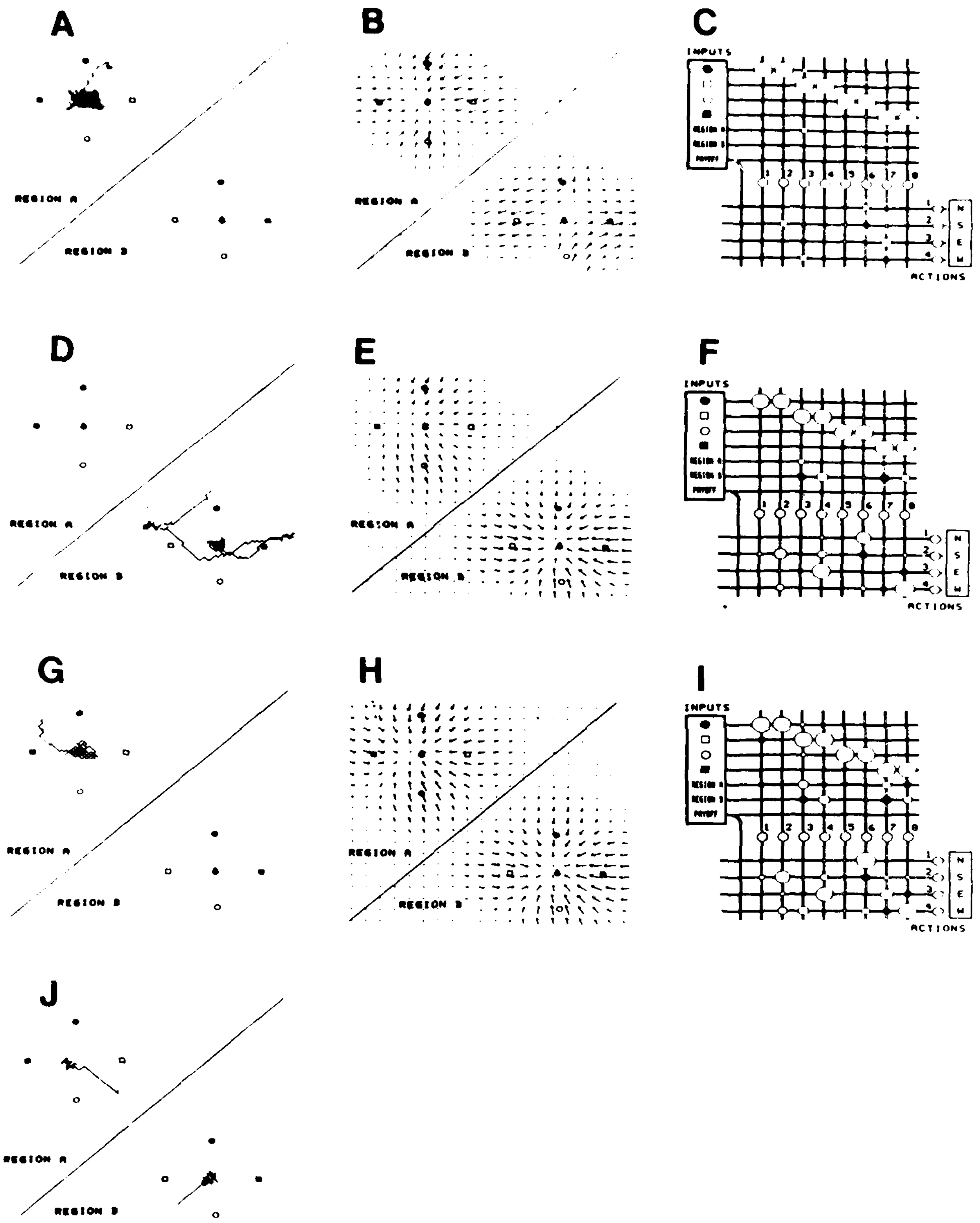


Figure 5

Solving the Two-Environment Landmark-Learning Problem.

Element 1: unused

Element 2: shaded circle in both regions

Element 3: hollow box in Region A

Element 4: hollow box in Region B

Element 5: unused

Element 6: hollow circle in both regions

Element 7: shaded box in Region A

Element 8: shaded box in Region B

Layer 2 can therefore generate the appropriate actions even though these actions are restricted to being linear functions of its input patterns. Although this process sounds complicated, it occurred with great reliability and was not particularly sensitive to parameter values.

Figure 5J shows the network behavior as the information was used that was stored during the experiences we have described. Both trees and their attractant distributions were removed, and the network started from places it had never before visited. Its path in each region shows direct approach to the former location of the tree.

The Creation of Landmarks

Layer 1 of the network described above might be regarded as havng "created" landmarks in the sense that it came to provide conjunctions of some of the original landmark signals with the region signals, that is, it created the landmark "hollow box in Region A," etc. The experiments we describe now were attempts to provide a more rich substrate of original signals out of which to create landmarks.

Figure 6 shows the structure of another two-layer network. The second layer of the network consists of four adaptive elements, each of which receives the reinforcement and the features r_1 through r_9 from the first layer and computes a component of the output vector (here, the output vectors are represented as $(+dx_1, -dx_1, +dx_2, -dx_2)$). Layer 1 of the network receives the reinforcement and the input vecto: (x_1, x_2) giving the coordinates of the network's current location in space, and computes the values r_1 through r_9, which are the components of the input vector to Layer-2. Each element of Layer 1 has the potential for representing a certain type of landmark.

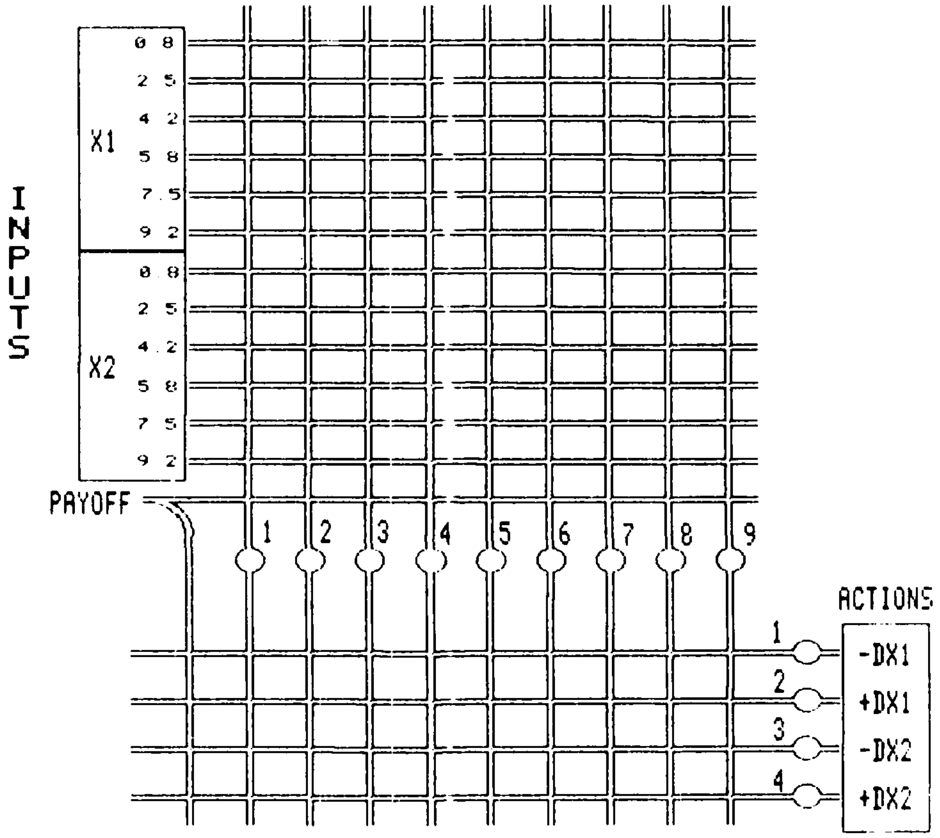


Figure 6

Two-Layer Network.

Each component of the input vector (x_1, x_2) is quantized into six intervals, and 12 variables are defined such that they are maximal at the center of unique overlapping intervals. Figure 7 shows the "receptive fields" of these 12 features (by the receptive field of a feature or element we mean the region of the input space for which the feature or element produces a nonzero signal). Given (x_1, x_2), the value of $q_i, i = 1, \ldots, 6$, is maximal when x_1 is the center of the interval corresponding to q_i; and the value of $q_i, i = 7, \ldots, 12$, is maximal when x_2 is the center of the interval corresponding to q_i. The values of the q_i decrease as the points x_1 (or x_2) move away from the center of the corresponding intervals (according to a Gaussian curve). The top six graphs in Figure 7 show the variables used to represent x_1, and the bottom three graphs show three of the six variables representing x_2. One can view these variables as providing the basis set of landmarks out of which others can be constructed.

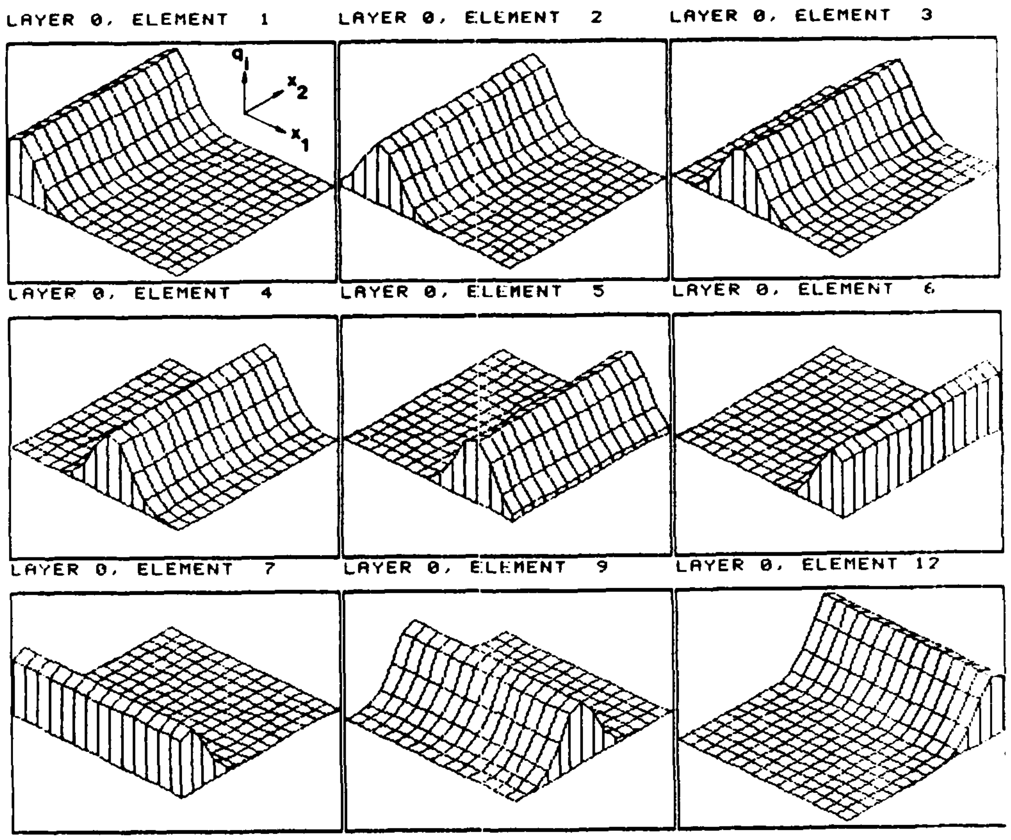


Figure 7

Features Initially Provided to the Network.

The elements comprising this network learn according to a rule similar to the one used in our other landmark-learning examples. Complete details are provided by Anderson (1982). An additional mechanism is employed, however, to prevent several Layer-1 elements from redundantly forming the same feature. A method for *enforcing variety* is needed. At each time step, the Layer-1 element with the largest output is selected as the one that is eligible for learning. The other elements produce outputs that influence Layer 2 but do not change their weights. Although this selection procedure can be implemented by well-known parallel, lateral-inhibitory mechanisms, we used standard programming to implement it (see Barto, Anderson, and Sutton, 1982, for further discussion and references).

Example 1 — We now describe the behavior of the system using a simple example. The reinforcement function z is defined as shown in Figure 8. This figure contains the same base plane as shown in Figure 7, but the height above the plane here depicts z, the reinforcement function, which has a maximum at position (5,5). The task of the reinforcement-learning system is to learn to generate in response to the input pattern generated at any position the action that results in the next position being as close to (5,5) as the set of possible actions permits. We set the position to (5,1) and ran the system for 700 steps. Figure 9 is a record of every fifth step, and the asterisk marks the final position. The system successfully learned what actions to associate with each position it visited. This can be seen in the vector-field display of Figure 10. Note the generalizaton to positions that had not been visited. This mapping was refined by randomly selecting 500 additional starting positions and running the system for one step in each. The resulting mapping appears in Figure 11.

Figure 12 shows the final receptive fields of the elements in Layer 1. Elements 1, 3, 6 and 9 encode the only significant features that formed. This is more noticeable in Figure 13. In this network display, the weights in Layer 2 show that only four Layer-1 elements contribute to the action-selecting process.

In addition to viewing the elements as becoming tuned to certain input situations, one can also view the elements as becoming tuned to certain compound output actions. Whereas the weights on the input side of an element determine its receptive field, the weights on its output side, through which its actions affect other elements, determine its "projective field." From Figure 13, we can determine that the features computed by

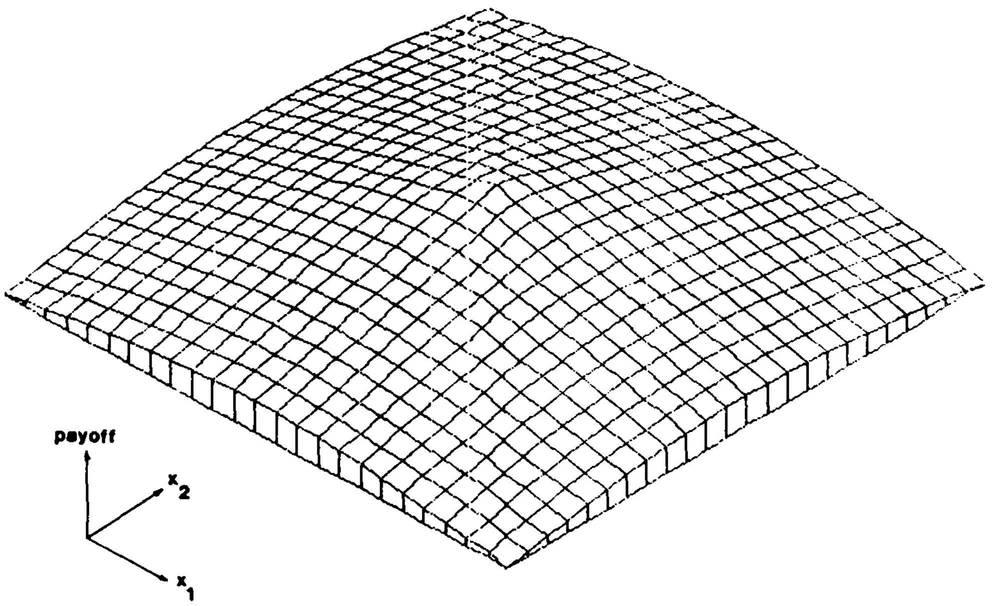


Figure 8
Example 1: The Payoff Function.

Elements 1, 3, 6 and 9 of Layer 1 are respectively associated with the actions $(+dx_1, -dx_2)$, $(-dx_1, +dx_2)$, $(-dx_1, -dx_2)$, and $(+dx_1, +dx_2)$. These elements have thus become very simple examples of "command cells" that trigger a complex of actions. If a system like this were being applied to the control of a two-jointed arm, for example, then activity in these elements might simultaneously increase the angular displacements of both joints (a very simple "synergy").

Example 2 — The purpose of this experiment was to demonstrate network performance on a task for which the layered structure of the network is necessary. The reinforcement function defining the task is shown in Figure 14. To obtain the largest reinforcement, the system should learn to generate actions that result in movement toward position (2.5,5), the payof function's maximum, and away from position (7.5,5), the function's minimum.

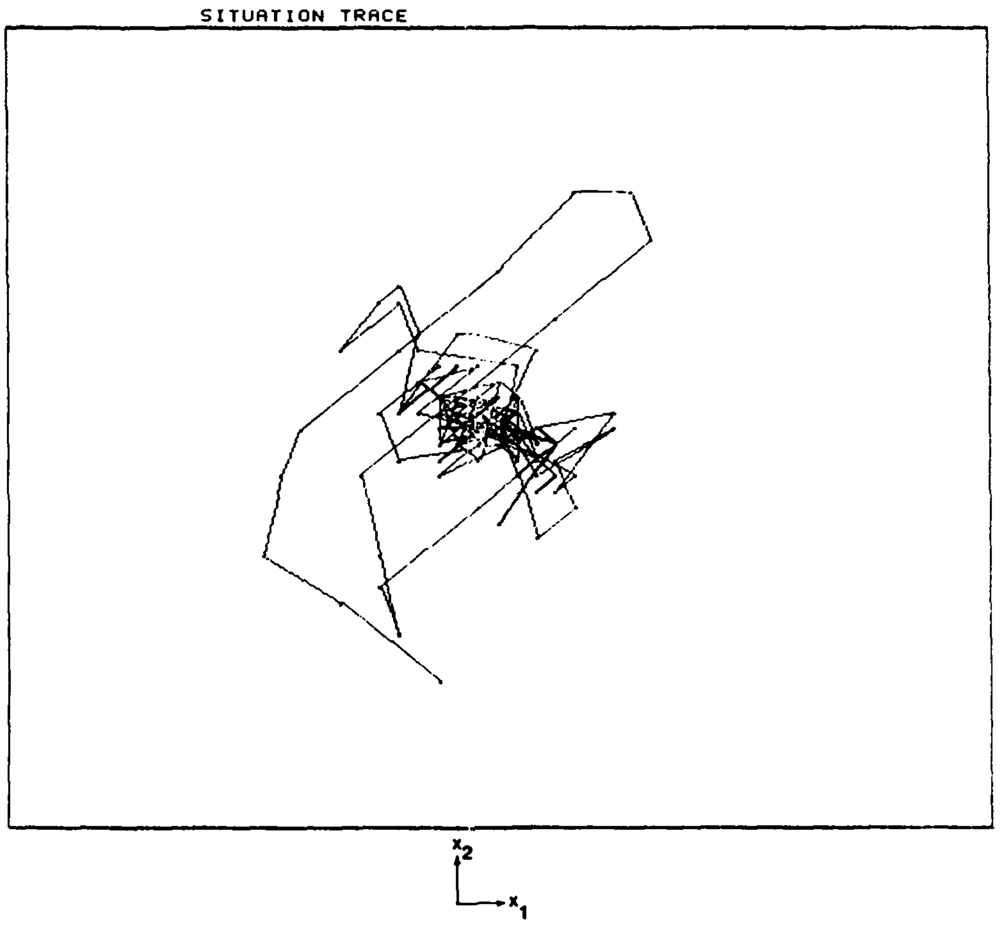


Figure 9

Example 1: Trace of the Situation Changes.

Anderson (1982) provides a proof that Layer 2 alone cannot solve this task, assuming that the positions are represented in terms of the q_i 's described above (like the two-environment problem discussed above, it is essentially an exclusive-or problem).

The training procedure for this experiment was slightly different from that used in the previous experiment. Since the system produced actions that rapidly carried it away from the area to be avoided, it was necessary to place the system periodically near this area in order to provide it with enough experience in this region. The system was trained for a total of 10,000 steps, with a random position change after every 20 steps. Most of the

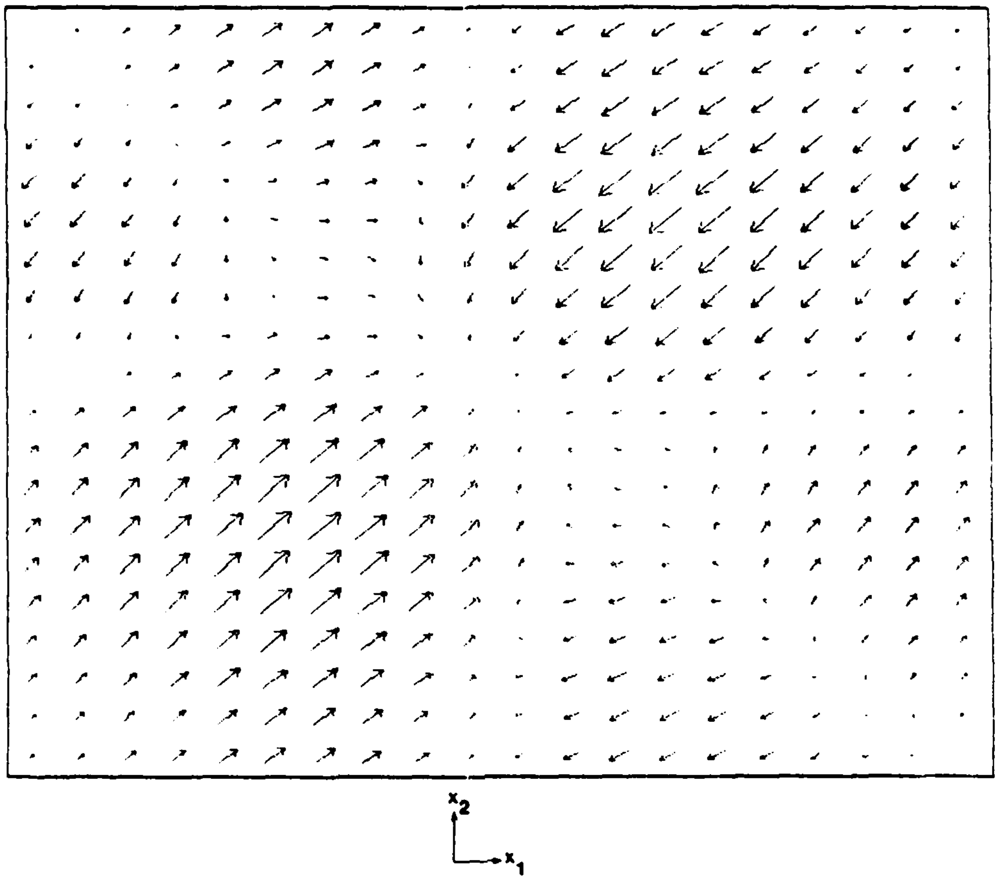


Figure 10

Experiment 1: Vector-Field Display after movement shown in Figure 9.

mapping was formed early in the experiment, but many additional steps were required for the mapping to be learned completely near the payoff function's minimum. The number of steps could have been reduced if the system had been selectively placed where more experience was necessary.

Figure 15 shows the resulting vector-field display. Figure 16 shows the final receptive fields of the Layer 1 elements. In addition, the receptive fields of the Layer 2 elements are shown in Figure 17. The dx_1 components of the actions for each position can be computed by subtracting the output of Layer 2's Element 1 from the output of Layer 2's Element 2.

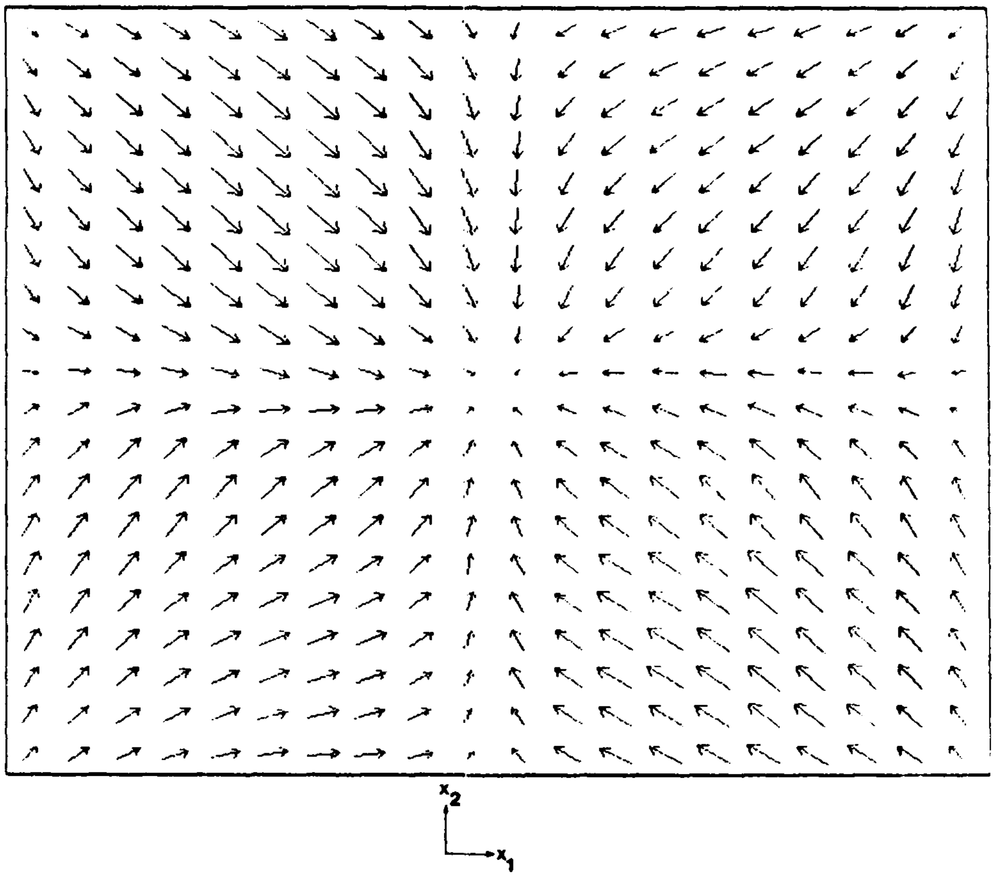


Figure 11

Vector Field-Display after an additional 500 steps.

Similarly, the dx_2 components of the actions can be computed by subtracting the output of Layer 2's Element 3 from the output of Layer 2' Element 4. Figure 18 shows the results of these subtractions. Notice that these functions approximate the partial derivatives of the reinforcement function, which is just what is desired.

Discussion

The simulation experiments described above represent initial explorations of the behavior of layered networks of goal-seeking components. Although the results of these experiments indicate that goal-seeking elements can effectively learn when embedded in

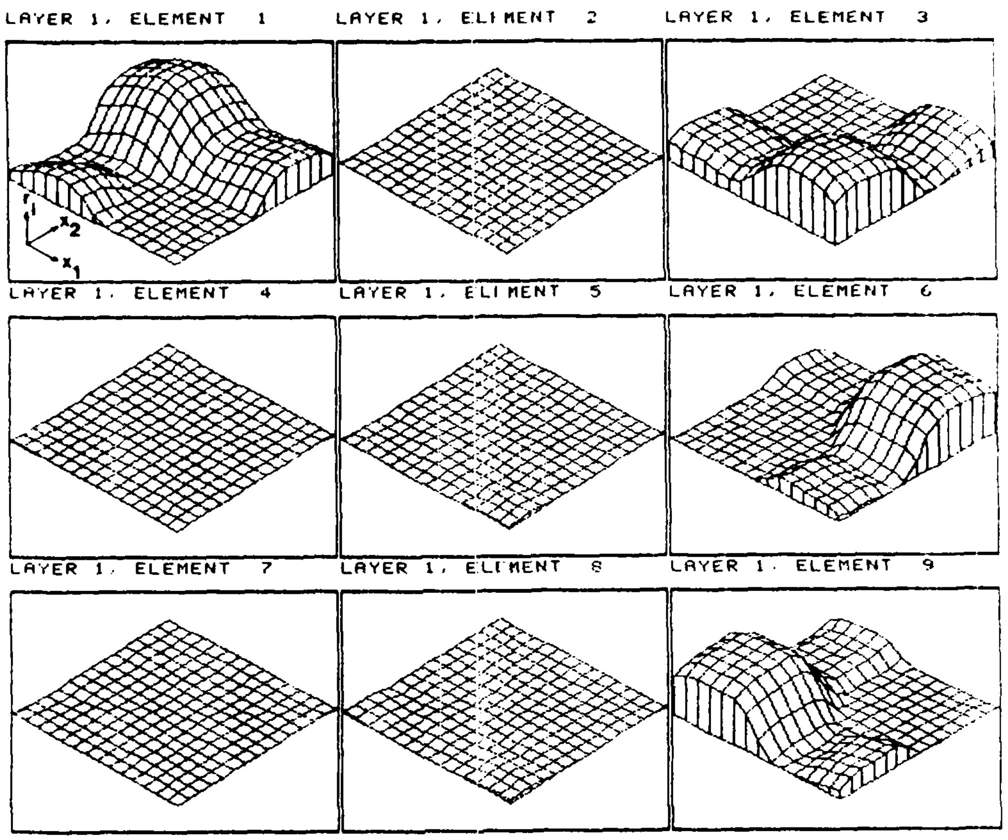


Figure 12

Example 1: Functions Computed by Layer-1 Elements.

networks, the experiments were too simple to allow us to conclude that we had solved the problem of learning nonlinear mappings by layered networks in an extensible way. In particular, the networks were not deep enough to allow us to determine if they could robustly implement a parallel version of a beam search. We regard these experiments as successful initial studies of networks of goal-seeking components.

Two kinds of questions are raised by our simulations. One concerns network architecture: How organized must the initial network be in order to support appropriate learning? The other kind of question concerns the individual adaptive elements: Do adaptive elements need more, or less, sophistication in their adaptive abilities?

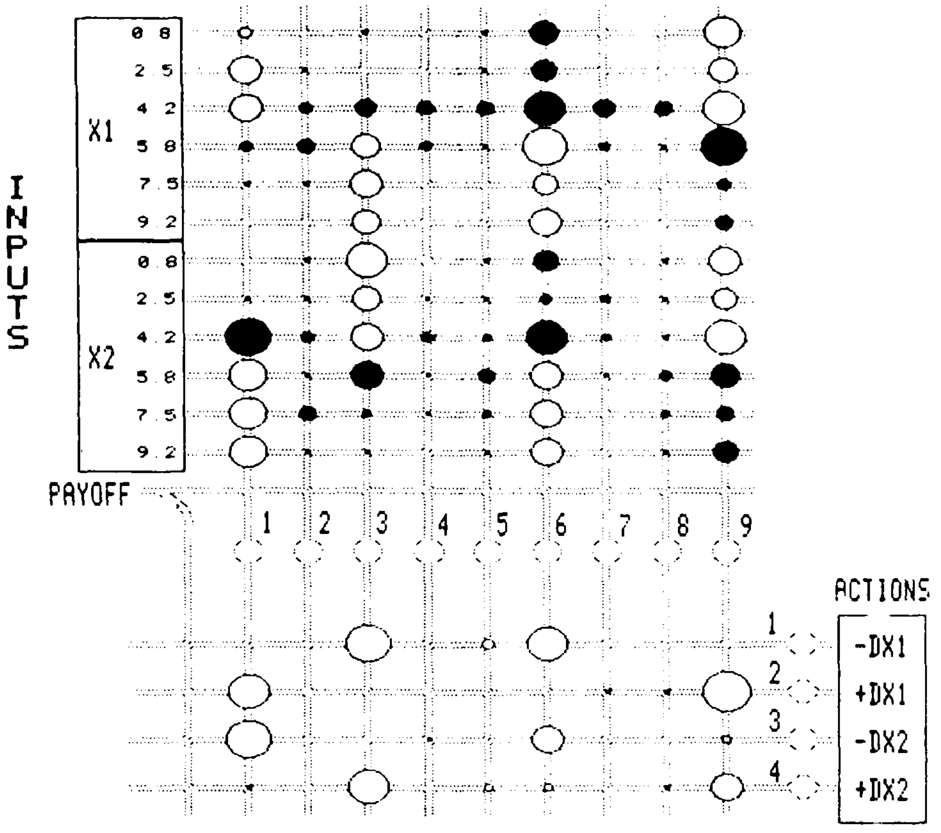


Figure 13

Example 1: Network Weights.

We indicated in Section 1 that our concern with networks of minimal *a priori* structure is not the result of any belief that learning begins with a *tabula rasa*. On the contrary, as much knowledge as is initially available ought to be brought to bear on any task. We are concerned with filling in the unspecified residue, which will be more or less depending on the task. Therefore, we would like our networks not to be initially structured in a strongly problem-dependent way. One of the difficulties with our two-environment illustration is that the initial network appears to have been "set up" exclusively for that problem. The Layer-1 elements were explicitly paired in order to "split" the landmark signals, and "Region A" and "Region B" signals were conveniently supplied.

Obviously, the network was indeed set up for the given problem, but its design incor-

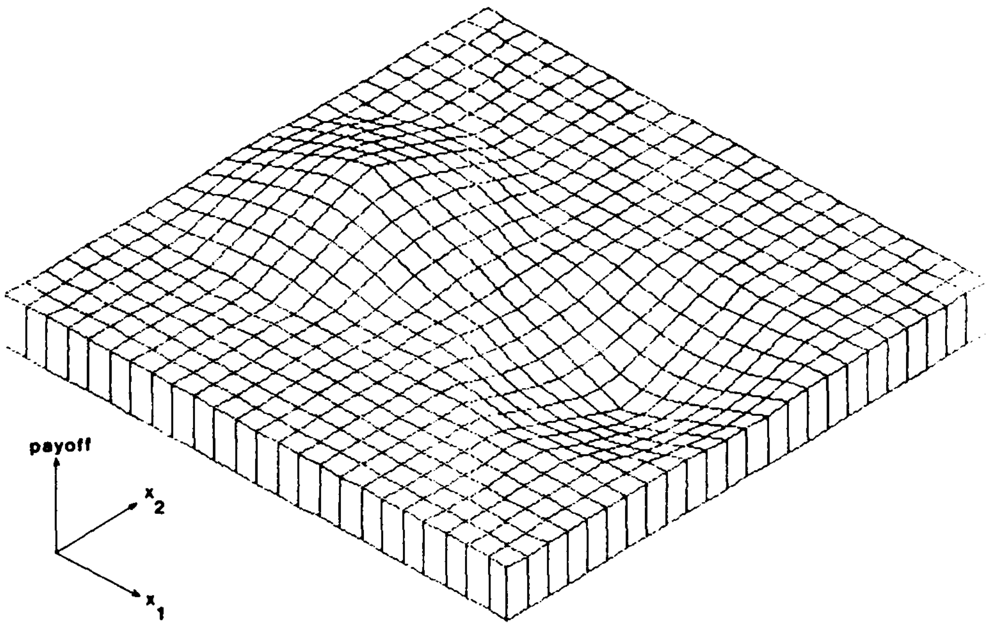


Figure 14

Example 2: Payoff Function.

porates some general principles that are obscure due to the simplicity of the problem. The "Region A" and "Region B" inputs do not play a specialized role in the learning equations of the Layer-1 elements. If it had been necessary for the problem's solution, the Layer-1 elements could have formed more complex discriminations based on several signals. We regard the "Region A" and "Region B" inputs as leading to the simplest case in which the regions can be distinguished by linearly separable patterns of input features. Although the existence of these single signals does make the problem easier, it does not trivialize it since the overall problem remains nonlinear.

The explicit pairing of the Layer-1 elements, where only the most strongly activated element of each pair learns, was used to enforce variety during the learning process. We used a similar device in the second set of experiments described in this section, and we described several other instances of this general idea in the above review of earlier research.

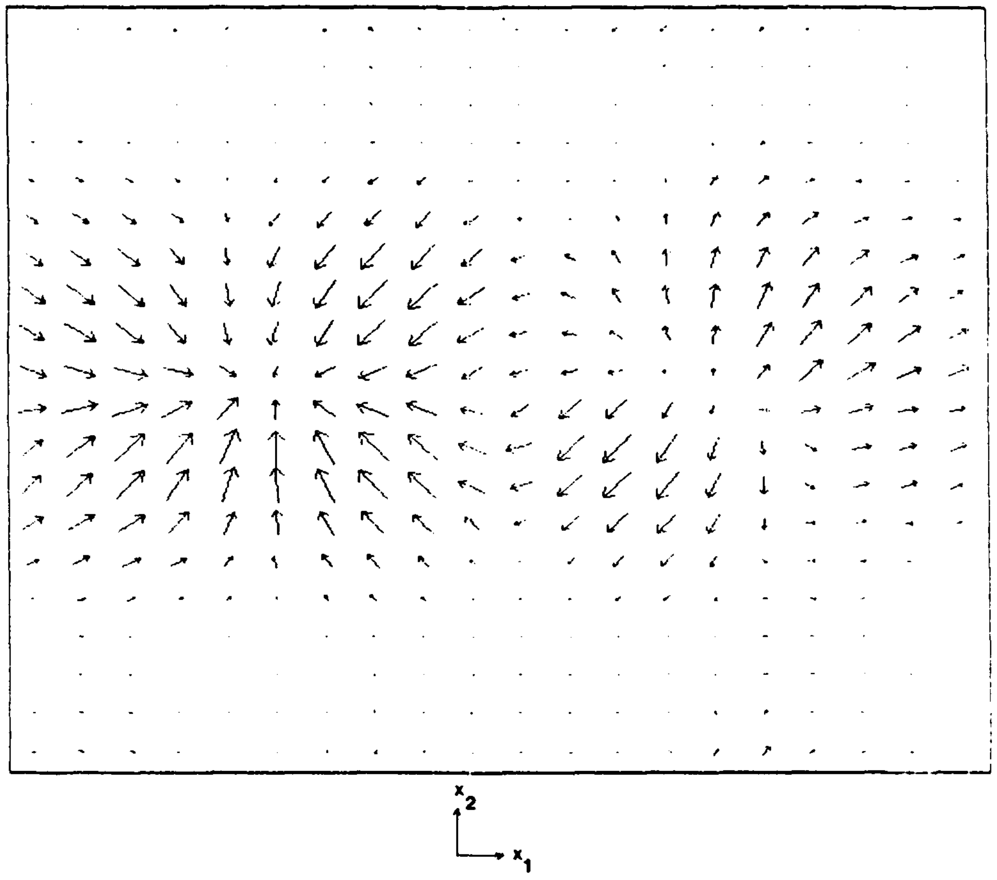


Figure 15

Example 2: Vector-Field Display.

We view the element pairings in the two-environment network as the simplest case of "laterally inhibitory" pools of elements. Consequently, the element pairs are not as *ad hoc* as they might at first appear.

A more serious deficiency in the design of the two-environment network is that the number of elements in Layer 1 is the same as would be required if we just initially supplied *all* of the possible conjunctions of landmark and region signals. In other words, in extending this type of network to larger problems, we would not be avoiding the combinatorial explosion of hardware units. There should be a way of permitting unused elements to enter

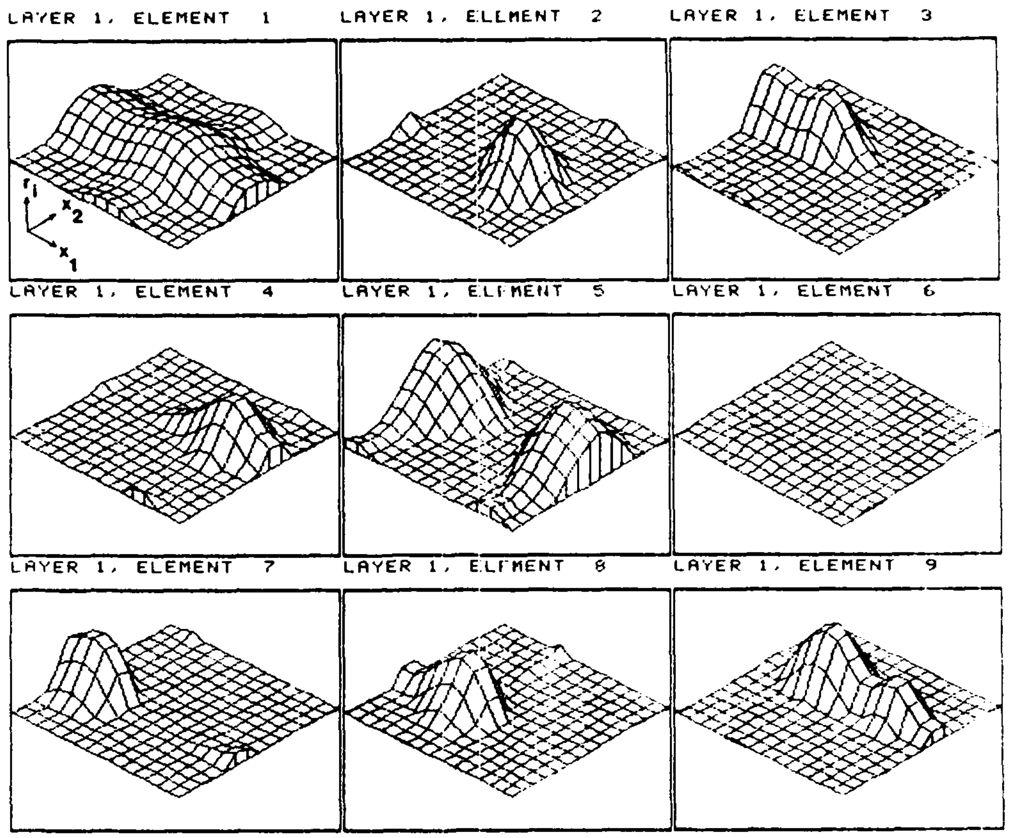


Figure 16

Example 2: Functions Computed by Layer-1 Elements.

into whatever combinations turn out to be useful for solving the problem. Ideally, perhaps, one would like an approach in which whatever hardware resources were available could be allocated in a reasonably unrestricted way based upon the demands of the task (although *complete* equipotentiality would not be necessary). In the two-environment network, on the other hand, an unused element of one of the pairings could not be reallocated to act, for example, as the third element of a three-way split of some other signal.

There are several approaches to solving this type of problem, some of which require the elements to have more sophisticated adaptive capabilities. For example, an element's weights might stabilize to the extent that the element is exerting an influence upon other elements. Elements that are not "listened to" should retune themselves to other features.

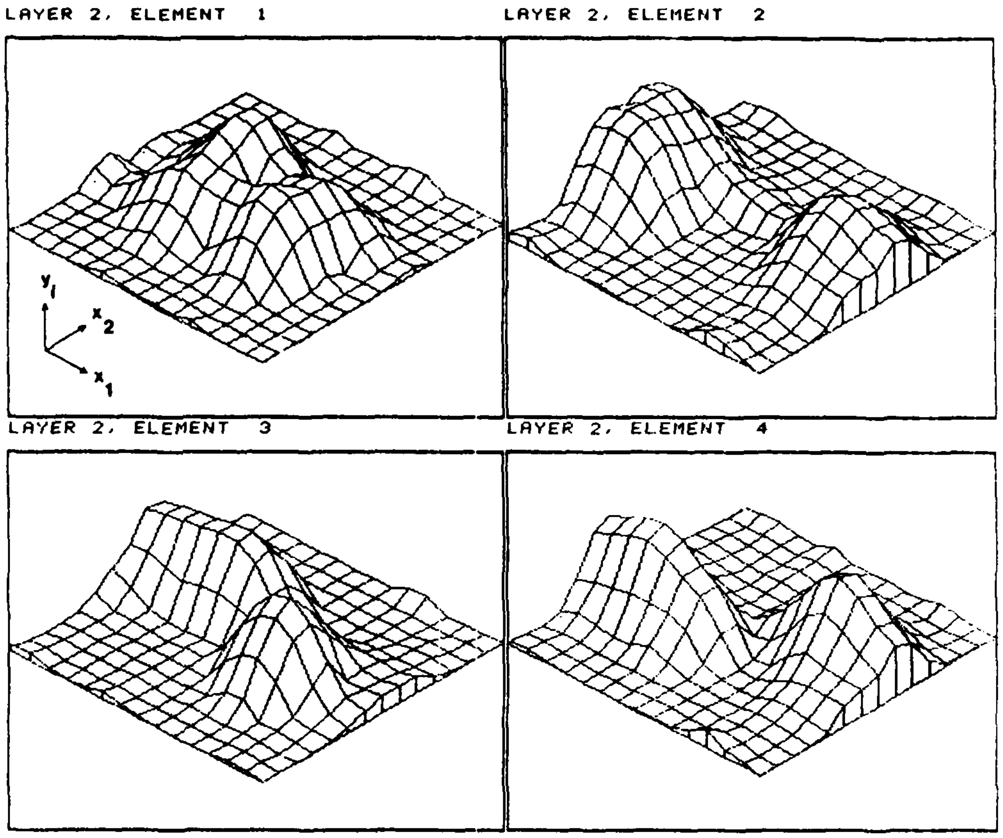


Figure 17

Example 2: Functions Computed by Layer-2 Elements.

This mechanism is similar to the dropping of elements of low worth as discussed above (Selfridge, 1959; Klopf and Gose, 1969). Elements could be literally dropped or they could just be "released" from their tuned state to start over again. We have not yet experimented with elements that monitor their output weights and use that information to alter their input weights, but we believe that these capabilities might be useful in layered networks.

Despite these shortcomings, the two-environment example does illustrate an advantage of not committing elements to higher-order features until these features are required. Unnecessarily "splitting" a variable prevents generalization from taking place along that dimension of the representation. This is illustrated by the example shown in Figure 5. After experience in Region A, the network attempted to use the relationships learned in Region

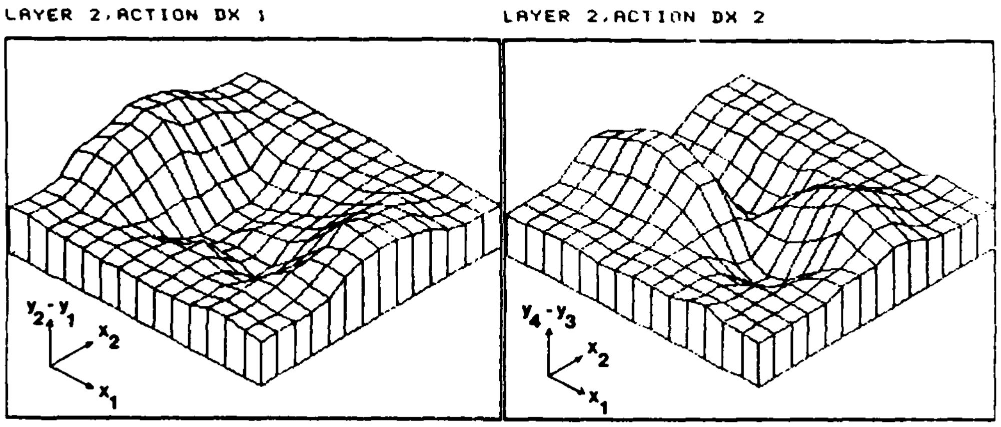


Figure 18

Example 2: Final Situation-to-Action Mapping.

A to guide its behavior in Region B. The relationships involving the boxes happened to be inappropriate in Region B, but those involving the circles successfully generalized. The network immediately moved south when placed in the northern part of Region B (Figure 5D). If separate variables for each landmark in each region had been supplied initially to the Layer-2 network, then no use of the Region A experience would have been attempted in Region B, and learning would have been slower.

Most importantly, all of the simulations described here show the formation of element coalitions. First and second layer elements cooperated in order to obtain higher levels of reinforcement than each element could obtain by acting alone. In this case, the coalitions took the form of the linked pathways through the networks. Nonzero weights from Layer-1 to Layer-2 elements implied that the elements were no longer statistically independent in their operation. For example, a large positive weight between Layer-1 Element A and Layer-2 Element B meant that when A fired, B almost surely fired as well — to the benefit of both elements.

Finally, we wish again to bring attention to the adaptive development of Layer-1 elements' "projective fields." Given adaptive Layer-1 elements (or, in general, adaptive interior elements), one can look at how they influence other elements as well as how they are themselves influenced. We pointed out that several elements in Figure 13 have become rudimentary "command cells" by becoming linked to compounds of primitive actions. The opportunity to observe the development of this type of organization is absent in cases in which only the network's output elements adapt. We think that this phenomenon will be a key feature in obtaining efficient search performances in problems more difficlt than those we have studied so far. The adaptive development of hierarchies of command cells provides a means for giving necessary structure to a network's search strategy.

Section 3
EXPERIMENTS WITH NONASSOCIATIVE PROBLEMS

Introduction

In this section we begin the description of a series of simulation experiments designed to help us better understand and refine the class of learning rules that we have studied. The first series of experiments concern nonassociative learning tasks in which the only signal the system receives from its environment is a scalar reinforcement signal. Such tasks are nonassociative because there is no non-reinforcing, or neutral, input with which to associate actions. We also refer to these tasks as *reinforcement-only tasks.* In these tasks learning system must discover and consistently choose an action to maximize some measure of cumulative reinforcement. These tasks allowed us to examine the relationship of our algorithms to well-studied classes of nonassociative learning algorithms. The experiments were designed to compare the convergence rates of a variety of algorithms across a variety of nonassociative tasks. Although ideally one would like to analytically determine the relative convergence rates of the various algorithms, in practice this is very difficult. We considered a large number of algorithms and tasks, and convergence rates are very difficult to obtain even in relatively simple cases. We are continuing, however, to pursue mathematical analyses of the behavior of certain algorithms.

Search Under Uncertainty

We pointed out in Section 2 that the optimization problem faced by an individual adaptive element that is deeply embedded in a network is characterized by a high degree of uncertainty. The reinforcement feedback received by such an element at any time will generally depend upon the actions of a large number of components taken at a variety of earlier times. Consequently, a single element's actions will contribute to its own reinforcement only in a statistical sense. Although a variety of measures need to be employed to reduce this uncertainty (for example, the use of an adaptive critic as discussed in Section 6), we do not believe that it is possible to reduce it to an inconsequential level.

The nonassociative tasks described here permitted us to focus upon search under uncertainty in the absence of the confounding influences introduced by the formation of associative mappings. These tasks also allowed us to draw upon the well-developed theory of search under uncertainty known as *learning automata theory* (LAT) in engineering (Tsetlin, 1973; Narendra and Thathachar, 1974; Lakshmivarahan, 1981) and *mathematical learning theory* (MLT) in psychology (Bush and Mosteller, 1955; Bush and Estes, 1959; Luce, Bush and Galanter, 1963, 1965; Atkinson, Bower and Crothers, 1965).

In LAT, an automaton interacts with an environment that evaluates its action in a probabilistic manner. If the automaton has n actions, then the environment is characterized by n reward probabilities, each of which specifies the probability that the automaton will receive a reward signal as a result of performing one of its possible actions. The optimal action is the one corresponding to the largest reward probability. Initially, no information is assumed about which action is optimal. In the case of a stochastic automaton, the automaton selects an action according to a probability function, receives the environment's response, and changes its action probabilities based upon that response. The next action is then selected according to the new action probabilities, and the process repeats. A stochastic automaton that improves performance, in the sense of increasing its expectation of reward, is called a *stochastic learning automaton* (Narendra and Thathachar, 1974). Note that due to the uncertainty in the feedback, even for the case of two actions this search task is highly nontrivial (whereas it would be trivial for deterministic environments). This type of problem has also been called an "n-armed bandit problem" (e.g., Bradt, et al., 1956; Cover and Hellman, 1970).

Several measures of performance have been proposed, and many algorithms have been studied. Stochastic learning automaton algorithms usually perform global search and therefore cannot become trapped on local optima. Unlike hill-climbing algorithms, for example, they do not search by trying actions that are near previous actions. Actions are not related to one another in terms of nearness, or in any other sense. Since they are not local gradient methods, stochastic learning automaton algorithms typically do not incorporate explicit comparisons of current reinforcement level with past reinforcement levels, as do hill-climbing algorithms. Within the stochastic learning automaton framework, it is not useful to estimate the change in reinforcement as a function of change in action. It may nevertheless be useful for such an algorithm to compare current reinforcement with past reinforcement levels. We call such algorithms *reinforcement-comparison algorithms*. Current learning automaton research is largely restricted to algorithms that are not reinforcement-comparison algorithms (which we call non-reinforcement-comparison algorithms). We have recently become aware, however, of several algorithms discussed by Mars and Poppelbaum (1981) that are reinforcement-comparison algorithms. We have not yet investigated the relationship between these algorithms and our own.

Most of the work on reinforcement-learning systems with neutral as well as reinforcing input has been done with non-reinforcement-comparison algorithms derived from those studied in LAT and MLT. This is the reason we were particularly interested in making a comparison between reinforcement-comparison and non-reinforcement-comparison algorithms. We hoped to establish whether or not reinforcement-comparison techniques improve performance. MLT and LAT have demonstrated that non-reinforcement comparison algorithms are capable of eventually solving reinforcement-only problems. Several forms of asymptotic optimality — optimal or near optimal performance of the algorithms as time goes to infinity — have been shown for the non-reinforcement-comparsion algorithms considered in these fields. However, in practice, rate of convergence is of critical importance, and proveably (near) optimal algorithms tend to be very slow in converging to optimal actions.

In this section we present the results of a series of computer simulations of 10 learning algorithms applied to 6 reinforcement-only tasks. The experiments were designed to compare the convergence rates of the algorithms across tasks.

Tasks

The 6 learning tasks are summarized in Table 1. There are 3 each of 2 types of tasks, called *binary-reinforcement tasks* and *continuous-reinforcement tasks*. In binary-reinforcement tasks each interaction of the learning system with the environment has one of two outcomes. The goal of the learning system is to maximize the number of interactions that end in one outcome, called *success*, and minimize the number that end in the other outcome, called *failure*. The learning system is made aware of the outcome of its action $y[t]$ taken at time t by the reinforcement $r[t+1]$ it receives at time $t+1$. If the outcome is success, $r[t+1]$ has the value +1, and if the outcome is failure, $r[t+1]$ has the value -1.

Table 1

Summary of Nonassociative Learning Tasks

Task	Reinforcement	r range	r mean		Relevant
Number	Type		Action 1	Action 0	Algorithms
1	Binary	$\{1,-1\}$	.9	.8	1-9
2	Binary	$\{1,-1\}$	.2	.1	1-9
3	Binary	$\{1,-1\}$	.55	.45	1-9
4	Continuous	$\Re$	.9	.8	4-10
5	Continuous	$\Re$	−.8	−.9	4-9
6	Continuous	$\Re$	.05	−.05	4-9

In continuous-reinforcement tasks the outcome of each interaction of the learning system with the environment is a continuous-valued reinforcement. For example, the reinforcement $r[t+1]$ corresponding to the action $y[t]$ might take on values in the real interval $[-1,1]$. In continuous-reinforcement tasks the goal of the learning system is to maximize the expected value of this reinforcement.

All 6 tasks are binary-action tasks. The learning system chooses one of two actions, denoted $y[t]=0$ and $y[t]=1$.

For binary-reinforcement tasks, the environment is fully characterized by two conditional probabilities. One is the probability that the reinforcement $r[t+1]$ is -1 given that the preceding action $y[t]$ was 0, and the other is the probability that the reinforcement $r[t+1]$ is $+1$ given that $y[t] = 1$. The success probabilities conditional on each action for the 3 binary-reinforcement tasks we studied are given in Columns 4 and 5 of the first three rows of Table 1.

For simplicity the tasks were constructed such that Action 1 is the better action on all tasks. On Tasks 1–3 this was achieved by selecting a higher success probability conditional on Action 1 than conditional on Action 0. These tasks include one on which both success probabilities are high (Task 1), one on which both success probabilities are low (Task 2), and one on which one of the success probabilities is greater than $\frac{1}{2}$ and the other is less than $\frac{1}{2}$ (Task 3).

One might expect a task such as Task 1 to present a special problem for algorithms because even the action with the lower probability of producing success is followed by success almost all (80%) of the time. On the other hand, a task such as Task 2 is problematic because the action with the higher probability of producing success is followed by failure almost all (80%) of the time. By contrast, a task such as Task 3 should be the easiest for algorithms to solve because the correct action is successful more than half the time and the incorrect action is punished more than half the time. Thus, these three tasks cover the major classes of non-trivial binary-reinforcement/binary-action tasks.

Columns 4 and 5 of the last three rows of Table 1 list the expected value of the reinforcement for each action on the three continuous-reinforcement tasks. For each of these tasks the reinforcement is selected according to a uniform distribution centered at the mean indicated in Table 1 and extending .1 to either side. For example, if the learning system selects Action 1 at time t on Task 4, then the reinforcement at time $t+1$ is selected according to a uniform distibution over the interval from 0.8 to 1.0.

Tasks 4–6 were also constructed so that Action 1 is the preferable action on all tasks. These three continuous-reinforcement tasks were selected to cover roughly the same space of relative difficulties as the binary-reinforcement tasks. One task was constructed so that the expected reinforcement would be positive for both actions (Task 4), one so that the expected reinforcement would be negative for both actions (Task 5), and one so that the expected reinforcement would be positive for one action and negative for the other (Task 6).

In addition, we wanted at least one task (Task 4) on which the reinforcement was limited to the real interval $[0, 1]$, because this is a condition required for the applicability of some algorithms. We selected a fairly narrow range for the uniform distribution used to select the reinforcement and selected the means of the distribution fairly close together because we felt that this would amplify the ways the binary-reinforcement and continuous-reinforcement tasks might differ from one another.

Algorithms

The 10 learning algorithms are summarized in Table 2. Each algorithm selects its actions probabalistically; $\pi[t]$ denotes the probability with which an algorithm chooses action $y[t] = 1$. Not all algorithms were applied to all tasks. The tasks to which each algorithm was applied is indicated in the last column of Table 2.

Algorithms 1–3 are well-known learning automata algorithms. They are, respectively, linear reward-inaction (L_{RI}), linear penalty-inaction (L_{PI}), and linear reward-penalty (L_{RP}). These algorithms apply different update equations for success and failure and are only applicable to the binary-reinforcement Tasks 1–3.

Algorithm 10 is also a learning automaton algorithm (Mason, 1973). It was designed for tasks in which the environment returns a real-valued reinforcement in the real interval $[0, 1]$. These tasks are called S-model environments in the learning automata literature (Narendra and Thatachar, 1974). Algorithm 10 was applied only to Task 4, because this was the only task in which the reinforcement was always between 0 and 1.

Whereas Algorithms 1–3 and 10 directly update their probability π of choosing Action 1, Algorithms 4–9 operate by updating a modifiable parameter w that determines this probability according to the unit normal distribution function $\Phi(\cdot)$ having mean w and standard deviation σ:

Table 2

Summary of the Nonassociative Learning Algorithms.

Number	Update Rule	Relevant Tasks
1	$\pi[t+1] = \pi[t] + \begin{cases} \alpha(y[t] - \pi[t]), & \text{if } r[t+1] = 1 \\ 0, & \text{if } r[t+1] = -1 \end{cases}$	1-3
2	$\pi[t+1] = \pi[t] + \begin{cases} 0, & \text{if } r[t+1] = -1 \\ \alpha(1 - y[t] - \pi[t]), & \text{if } r[t+1] = 1 \end{cases}$	1-3
3	$\pi[t+1] = \pi[t] + \begin{cases} \alpha(y[t] - \pi[t]), & \text{if } r[t+1] = 1 \\ \alpha(1 - y[t] - \pi[t]), & \text{if } r[t+1] = 1 \end{cases}$	1-3
4	$w[t+1] = w[t] + \alpha r[t+1](y[t] - 1/2)$	1-6
5	$w[t+1] = w[t] + \alpha r[t+1](y[t] - \pi[t])$	1-6
6	$w[t+1] = w[t] + \alpha(r[t+1] - r[t])(y[t] - 1/2)$	3,4
7	$w[t+1] = w[t] + \alpha(r[t+1] - r[t])(y[t] - \pi[t])$	3,4
8	$w[t+1] = w[t] + \alpha(r[t+1] - p[t])(y[t] - 1/2)$	1-6
9	$w[t+1] = w[t] + \alpha(r[t+1] - p[t])(y[t] - \pi[t])$	1-6
10	$\pi[t+1] = \pi[t] + \alpha r[t+1](y[t] - \pi[t])$	4

Where:

$$w[0] = 0 \qquad v[0] = 0 \qquad y[t] \in \{1, 0\} \qquad \alpha > 0 \qquad \pi[0] = 1/2$$

$\pi[t]$ is the probability that $y[t] = 1$

For Algorithms 4-9:

$$\pi[t] = \Phi(w[t]/\sigma)$$

where $\sigma = 0.3$ and $\Phi(\cdot)$ is the unit normal distribution function.

For Algorithms 8 and 9:

$$p[t+1] = p[t] + \beta(r[t+1] - p[t])$$

where $p[0] = r[1]$ and $\beta = 0.2$

$$\pi[t] = \Phi(w/\sigma), \tag{1}$$

where $\sigma = 0.3$. Because there is no explicit mathematical form for the unit normal distribution function, Φ was approximated by table lookup and linear interpolation.

Computationally, Algorithms 4–9 determine their action $y[t]$ according to the sign of the sum of $w[t]$ and a random variable $n[t]$ selected according to a mean 0 normal distribution (with standard deviation $\sigma = 0.3$). If the sum is greater than 0, the action $y[t] = 1$ is chosen, otherwise the action $y[t] = 0$ is chosen:

$$y[t] = \begin{cases} 1, & \text{if } w[t] + n[t] > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

This manner of selecting actions yields the action probability as a function of w given by (1).

According to (1), any value of $w[t]$ determines a legitimate value for $\pi[t]$, i.e., one that does not exceed the range of allowable probabilities from 0 to 1. Algorithms that update $w[t]$, such as Algorithms 4–9, are therefore relatively easily applied to a wide range of different tasks. Unlike the algorithms that update $\pi[t]$ directly, one need never worry about $\pi[t]$ exceeding the allowed range of a probability. Algorithms 4–9 are the only algorithms applied to all 6 tasks.

Algorithms 4–9 include both reinforcement-comparison algorithms (Algorithms 6–9) and non-reinforcement-comparison agloriths (Algorithms 4–5) of several different types. Their differences and similarities will be brought out in detail when the results of the simulation experiments are discussed.

Each algorithm is parameterized by a single number α, called the learning constant, that determines the learning rate of the algorithm. For small positive values of α, learning is slow. For larger values of α, learning is faster, but if the value of α is too large, an algorithm can learn too quickly and converge to the wrong action.

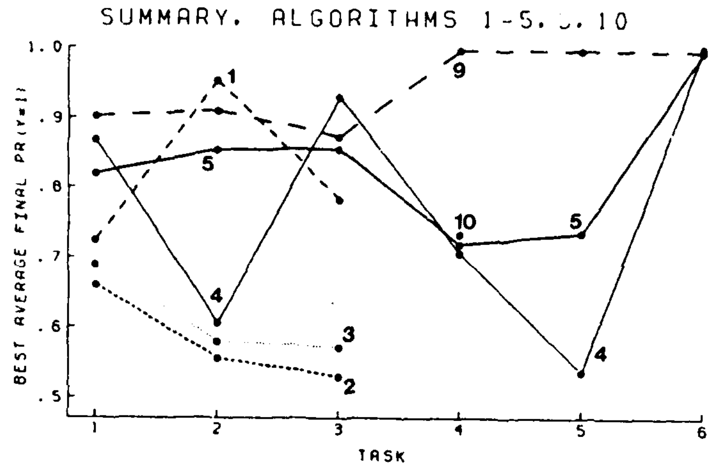
Results

For each algorithm and each task to which it was applied, simulation experiments were performed with 8 different values of α. These 8 values were the negative powers of 2, 2^0 through 2^{-7}, i.e., $\alpha = 1., .5, .25, .125, .0625, .03125, .015625,$ and $.0078125$. For each algorithm, task, and value of α, 200 simulation runs were made, each of 200 time steps. The 200 runs differ from each other only in the choice of the initial seed for the pseudorandom number generator.

At the end of each run the final probability with which the algorithm selected Action 1, $\pi[201]$, was determined. Since for all tasks Action 1 is the better of the two actions, this probability provides a measure of the performance of the algorithm. In most cases performance of each algorithm is an inverted-U shaped function of α. Since many of these algorithms are quite different and use α in different ways, it is not valid to compare the performance of different algorithms at the same α values. These results are best interpreted by comparing the performance level the algorithms attained with their own *best* values for α. Using this performance measure, Figure 19 compares all algorithms on all tasks.

One can see that most algorithms on most tasks did not reach probabilities near unity in 200 time steps. We chose run durations short enough so that the probabilities achieved by the end of a run reflected the rate of learning. The data shown in Figure 19 say nothing about the probabilities that might have been reached if the runs had been allowed to continue. Although such asymptotic behavior is important in understanding the mathematical structure of the algorithms, in practice learning rate is often more critical. Fast algorithms that sometimes do not converge can be more useful than slower algorithms that are asymptotically optimal.

Consider the lower graph of Figure 19. On Tasks 4–6, the performances of Algorithms 6–9 were virtually identical. On Tasks 1–3, Algorithm 9 was slightly better in each case, but the general trends in performance as α varied were very similar, and the best performances of these algorithms were not greatly different. Since these algorithms performed so similarly, and since Algorithm 9 was the best of them in all cases, only data from Algorithm



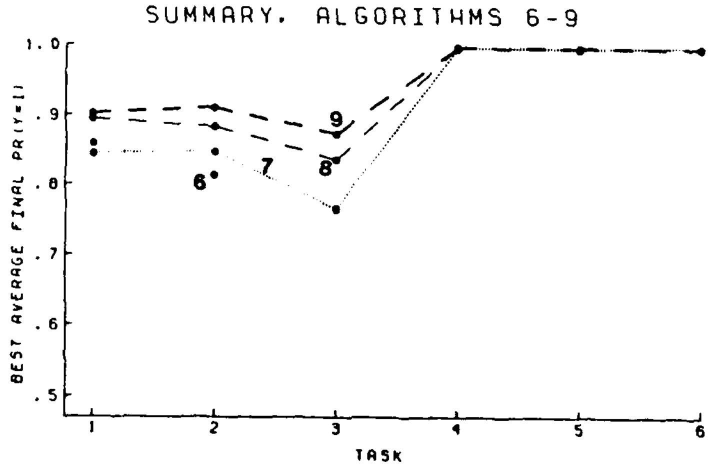


Figure 19

Summary of Algorithm Performance on all tasks of Section 3.

9 is plotted in the upper graphs of each figure for comparison with the other algorithms.

The performance plots in the upper graph of Figure 19 show much wider variation than those in the lower graphs. On all tasks except Task 6, some algorithms showed high performance whereas others showed very poor performance, and which algorithms were which varied from task to task. Algorithm 4, for example, performed the best of all algorithms on Tasks 3 and 6, but worst of all algorithms on Tasks 4 and 5, as well as performing very poorly on Task 2. Algorithm 1, on the other hand, performed the best of all algorithms on Task 2 and performed only reasonably well on Tasks 1 and 3, the only other tasks to which it was applied.

Of those algorithms whose performances are plotted in the upper graphs of Figure 19, the only one that performed well on all tasks was Algorithm 9. Algorithm 9 performed the best of all algorithms on Tasks 1, 4, and 5, tied for best on Task 6, and was a close second on the remaining Tasks 2 and 3. Almost all the performance differences between algorithms shown in this figure are highly statistically significant. For example, even the difference between the best performances of Algorithms 9 and 5 on Task 3, one of the smallest of the differences mentioned above, is statistically significant at the P=0.01 level.

Discussion

Algorithms 6–9: Short-Term vs. Long-Term Reinforcement Comparison — Algorithms 6–9 are reinforcement-comparison algorithms. Algorithms 6 and 7 compare reinforcement by computing the difference between the current reinforcement and the preceding reinforcement, $r[t+1] - r[t]$. Algorithms 8 and 9 are somewhat more complex. Rather than comparing current reinforcement with the immediately preceding reinforcement, they compare it with a long-term measure of prior reinforcement. These algorithms compare current reinforcement with a real-valued variable, which we will call the *expected level of reinforcement*, whose value at time t is denoted $p[t]$. That is, they update w according to $r[t+1] - p[t]$, where $p[t]$ is an exponentially weighted average, or trace, of the preceding reinforcement values $r[t]$, $r[t-1]$, $r[t-2]$, ..., with the greater weight going to the more recent reinforcement values. Algorithms 8 and 9 use the following difference equation to produce this trace:

$$p[t+1] = p[t] + \beta(r[t+1] - p[t]), \tag{3}$$

where $p[0] = r[t]$ and $0 \leq \beta \leq 1$. β was 0.2 for all simulation runs reported here. Since $p[t]$ is incremented fractionally according to its difference from $r[t+1]$, $p[t]$ tracks the level of $r[t+1]$. If r remains constant, p eventually reaches that constant value. The parameter β determines how quickly p reaches a fixed level of r or how quickly it tracks a changing level of r. A fairly low value for β, such as the value 0.2 used in Algorithms

8 and 9, implies that $p[t]$ tracks $r[t+1]$ relatively slowly.

The lower part of Figure 19 shows that algorithms that compare current reinforcement with the longer term measure of preceding reinforcement (Algorithms 8 and 9) performed significantly better than the other two algorithms (Algorithms 6 and 7) on Tasks 1–3, and identically on Tasks 4–6.

Algorithms 4–9 — For the moment let us consider Algorithms 4–9 in pairs, 4 with 5, 6 with 7, and 8 with 9. The algorithms of each pair are identical except for the factors of their update equations involving which action was taken. The lower numbered algorithms of each pair use $y[t] - \frac{1}{2}$ in this capacity whereas the higher numbered algorithms of each pair use $y[t] - \pi[t]$. Both factors involve the action taken at time t, and the second factor uses knowledge of the action probability.

The factor $y[t] - \frac{1}{2}$ is $\frac{1}{2}$ when Action 1 is chosen and $-\frac{1}{2}$ when Action 0 is chosen. The absolute value of this factor is constant and its sign encodes which action was taken. The factor $y[t] - \pi[t]$ is somewhat more complex. On the earliest trials before $w[t]$ has changed very much from 0, this factor operates exactly as the factor $y[t] - \frac{1}{2}$ because $\pi[t]$ is still near its initial value of $\frac{1}{2}$. As $w[t]$ moves significantly away from 0, $\pi[t]$ moves toward either 0 or 1, and $y[t] - \pi[t]$ begins to behave differently from $y[t] - \frac{1}{2}$. The two factors always have the same sign, but their absolute values differ. In comparison to the factor $y[t] - \frac{1}{2}$, $y[t] - \pi[t]$ amplifies the changes in $w[t]$ on those steps on which the less-likely action is taken, and diminishes them on those steps on which the more-likely action is taken. For example, if $\pi[t]$ is .9, then $y[t] - \pi[t]$ is $.1 = 1 - .9$ if the more-likely action, Action 1, is chosen, and $-.9 = 0 - .9$ if the less-likely action, Action 0, is chosen.

It is not clear from the results shown in Figure 19 which of the algorithms of each pair is better. Algorithm 9, an algorithm using the factor $y[t] - \pi[t]$, performed somewhat better than Algorithm 8 on Tasks 1-3, and essentially equivalently to Algorithm 8 on Tasks 4–6. Algorithm 7, an algorithm using the factor $y[t] - \pi[t]$, was slightly better than Algorithm 6 on Task 2, worse on Task 1, and equivalent on Tasks 3–6.

The differences between the performances of Algorithms 4 and 5 were much larger and more varied than the differences between the performances of Algorithms 6 and 7 and Algorithms 8 and 9. The performances of Algorithms 4 and 5 were nearly the same on Tasks 4 and 6 (the difference on Task 4 was not statistically significant; the difference on Task 6 was significant, but it probably would not have occurred if higher values of α had been tried for Algorithm 5). On the remaining tasks, Algorithm 4 was significantly better than Algorithm 5 on Tasks 1 and 3, and dramatically worse on Tasks 2 and 5.

The reason for the very poor performance of Algorithm 4 on Task 2 is that Task 2 was the binary-reinforcement task in which the outcome was failure on almost all trials irrespective of the action chosen, although failure was slightly less likely (.8 versus .9) when Action 1 was chosen. Algorithm 4 uses the following update rule:

$$w[t+1] = w[t] + \alpha\, r[t+1](y[t] - 1/2), \tag{4}$$

where $r[t+1] \in \{-1, 1\}$ on Task 2. This algorithm moved w in the correct direction (positive) on Task 2, but as it started to favor Action 1, it began to slow down and ultimately stop. The problem was that as the algorithm started to choose Action 1 more often than Action 0, most failures started to occur on Action 1 trials simply because Action 1 trials were occurring more frequently, and most trials ended in failure in any case. The equilibrium probability of Algorithm 4 choosing Action 1 — that at which the expected value of $w[t+1]$ given $w[t]$ is the same as $w[t]$ — on Task 2 can be shown mathematically to be $4/7 \approx .57$ (see Sutton, forthcoming, for details). Figure 19 shows that the final probability of Algorithm 4 choosing Action 1 on Task 2 was near this value (it was in fact near this value for all values of α that were tried).

Algorithm 5 differs from Algorithm 4 only in the replacement of the factor $y[t] - \frac{1}{2}$ by $y[t] - \pi[t]$, and this is enough to prevent the poor performance exhibited by Algorithm 4. As Algorithm 5 started to choose Action 1 more often, it experienced more trials on which it chose Action 1 (whose outcome was failure), but because of the factor $y[t] - \pi[t]$, the effect on the probability of choosing Action 1 of each of these failures was diminished. Unlike Algorithm 4, Algorithm 5 continued to move the probability of choosing Action 1 toward 1.

In conclusion, these results show that algorithms with the factor $y[t] - \pi[t]$ sometimes show improved and sometimes degraded performance over the corresponding algorithms with the factor $y[t] - \frac{1}{2}$. Where the factor $y[t] - \pi[t]$ improves performance, it sometimes improves it dramatically, and where it degrades performance, it degrades it relatively little. Finally, for the best performing algorithms (8 and 9), the factor $y[t] - \pi[t]$ resulted in a small consistent improvement across tasks.

Algorithms 1–10: Reinforcement-Comparison Mechanisms —The algorithms with reinforcement-comparison mechanisms (Algorithms 6–9) performed much better accross tasks than the comparable algorithms without reinforcement-comparison mechanisms (Algorithms 4 and 5). Although Algorithm 4 performed best of all algorithms by small margins on Tasks 3 and 6, it performed far worse than the reinforcement-comparison algorithms on Tasks 2, 4, and 5. On Tasks 4 and 5 all reinforcement-comparison algorithms showed complete and correct learning, and all other applicable algorithms showed only a very low level of learning.

Algorithms 1, 2, 3, and 10 are learning automata algorithms commonly discussed in the literature. Although these algorithms are not particularly noted for their speed of learning, they provide useful reference points against which to assess the performance of the other algorithms. Algorithms 1–3 could be applied only to the binary-reinforcement tasks (1–3). Algorithms 2 and 3 were the poorest performers on these tasks, and Algorithm 1, on balance, performed significantly worse than several of the other algorithms, including particularly the best reinforcement-comparison algorithm, Algorithm 9. Algorithm 10 was the only learning automaton algorithm applied to Task 4. However, it did little better than the other algorithms in challenging the greatly superior performance of the reinforcement-comparison algorithms on this continuous-reinforcement task.

Conclusion

These results do not prove that reinforcement-comparison mechanisms are necessary to improve learning rate. Strictly speaking, the conclusion that reinforcement-comparison mechanisms greatly improve rates of learning applies only to this particular set of algorithms and tasks. However, these results suggest that remembering past reinforcement levels for subsequent comparison may be an essential feature of robust and high-performance learning algorithms for discrete-action nonassociative reinforcement-learning tasks.

The simulation experiments reported here need to be extended and verified in several ways. First, a great many more algorithms should be tried, including other algorithms from the learning automata literature, algorithms from the 2-armed bandit literature, and reinforcement-comparison algorithms other than those tried here. For example, the algorithm that forms separate estimates of the expected reinforcement for each action and then compares them to determine its action probability is a reinforcement-comparison algorithm, though different from those tested here, and would make an interesting addition to this study. Second, mathematical analysis of the asymptotic properties of various reinforcement-comparison algorithms needs to be developed in order to more fully understand them. Some preliminary analyses can be found in Sutton (forthcoming).

The experiments reported here should also be extended to other tasks. Other variants of the binary-action with binary or continuous-reinforcement tasks that were studied here should be tried, but it is probably more important to experiment with other classes of tasks. Tasks with more than 2 actions and/or more than 2 discrete reinforcement levels (called Q-model environments in the LAT literature) come to mind.

Section 4
EXPERIMENTS WITH ASSOCIATIVE-LEARNING PROBLEMS

Introduction

Associative-learning problems are problems in which a mapping from input to output must be formed by the learning system. Stimuli and actions must be associated in such a way that the occurrence of a stimulus evokes the most suitable or appropriate action. Learning problems in pattern classification, for example, are associative because associations are learned between patterns (stimuli) and classifications (actions). Reinforcement-only problems are not associative-learning problems because the learning system need only discover the best action; it need not associate it with any stimulus. However, pattern-classification learning problems, while they are associative-learning problems, are not typically reinforcement-learning problems because they do not require the learning system to discover the best classifications — these are provided by a "teacher" during a training sequence.

Independent-Step Associative Learning

An independent-step learning problem is one in which the interaction between the learning system and its environment can be broken up into single steps, each of which is independent of the other. For associative reinforcement learning, each step involves the presentation of one stimulus vector to the learning system, its selection of one action, and finally the delivery of a reinforcement feedback signal by the environment. Such steps are said to be independent if the stimulus, action, and reinforcement of one step do not influence the stimulus, action, and reinforcement of any other step. In particular, the action selected on one step may influence the reinforcement on that step but not that of any other step.

Formally, the restriction of independent-step learning problems means that the environment does not change state. For these problems we can characterize the environment by a sequence of stimuli, $x[1]$, $x[2]$, ..., and a map r such that

$$r: X \times Y \rightarrow \Re$$
$$r(x[t], y[t]) \mapsto r[t+1],$$

where X is the stimulus space, Y is the action space, and the reinforcement space $\Re$ is the set of real numbers. As usual, r may be stochastic. A single step involves the stimulus at time t, $x[t]$, the action at time t, $y[t]$, and the reinforcement at time $t+1$, $r[t+1]$. Note that the reinforcement is denoted as occurring one time step later than the stimulus and the action. We have chosen this convention because it is consistent with later descriptions of learning problems that cannot be broken up into independent steps.

In this section, we compare the performance of 11 associative-learning algorithms on 12 independent-step associative reinforcement learning tasks. The algorithms are adaptations of those studied in Section 3. The experiments were designed to compare the learning rates of these algorithms as functions of characteristics of the tasks. Among other things, these experiments allowed us to test whether the advantage of reinforcement-comparison algorithms in reinforcement-only tasks carry over to associative-learning tasks. Before discussing these experiments, we first review some different approaches to associative learning.

Approaches to Associative Learning

Over the years, in various fields, there have been at least three approaches to the problem of associating actions with stimuli: the *independent-associations approach*, the *stimulus-sampling-theory approach*, and the *linear-mapping approach*. We discuss each of these approaches to associative learning and the relationships between them.

An important issue in evaluating a particular approach to associative learning is how it handles stimulus similarity, since this determines when generalization between stimuli occurs and when discrimination is possible. *Generalization* between stimuli occurs when training to one stimulus transfers and influences behavior (action selections) to another stimulus. When the transfer is positive, i.e., when the action learned in the presence of the first stimulus becomes more likely to occur in the presence of the second stimulus as a result of the training, then the two stimuli are said to be similar. Stimulus *discrimination* refers to the ability of a learning system to perform two different actions in response to two different stimuli. In stimulus discrimination, one is usually concerned with discriminating beteen two stimuli that are similar, in which case generalization between them may make their discrimination difficult. The ability to *completely discriminate* between two stimuli implies the ability to overcome any generalization between them and behave differently in response to them with 100% reliability.

The Independent-Associations Approach — One approach to associative learning is to make a separate association to an action for each stimulus presented to the learning system. Since each association is distinct and separate from any other, a separate memory variable (or variables) is required for each stimulus. This requirement restricts the usefulness of this approach to problems with small numbers of distinguishable stimuli. The independent-associations approach has been used in reinforcement-learning control theory (Mendel and Mclaren, 1970), in mathematical learning theory (Bush and Mosteller, 1955), and occasionally used in learning automata theory (Lakshmivarahan, 1981; Witten, 1977) and artificial intelligence (Michie and Chambers, 1968a, b; see Section 6). It is also the basis of "table-lookup" approaches to storing mappings (Raibert, 1978). Because all stimuli are treated separately, no stimuli are similar and no generalization can occur. Stimulus discrimination is trivial because it is never necessary to discriminate between similar stimuli.

In short, the independent-associations approach ignores the problems of stimulus similarity, generalization, and discrimination. By ignoring these issues, the independent-associations approach gives up the opportunities they provide. Assuming similar stimuli should elicit similar actions, transfer of training due to generalization can greatly speed up learning. In "real life" learning problems it may be rare for any detailed stimulus to occur twice in the lifetime of the learning system. In such cases, generalization between stimuli is essential.

On the other hand, one advantage of the independent-associations approach is that it permits one to apply algorithms known to work in nonassociative learning problems directly to associative problems. To do this, one allots a separate instance of a nonassociative learning algorithm, such as one of those studied in Section 3, *for each stimulus* *that will ever be presented to the learning system.* Whenever a stimulus occurs, the corresponding algorithm takes one step. This reduces the associative-learning problem to a set of independent nonassociative learning problems. Using the independent-associations approach and this technique, any algorithm known to work for nonassociative learning problems can be applied with confidence to associative-learning problems. For this reason the independent-associations approach has remained of considerable theoretical interest despite its requirement for small numbers of distinct stimuli and its lack of generalization capabilities.

The Stimulus-Sampling-Theory Approach — One approach to associative learning that does include stimulus similarity and generalization is stimulus sampling theory (Estes, 1950). In stimulus sampling theory (SST), stimuli are represented as subsets of a large number of independently variable stimulus components. (In SST these are usually called stimulus elements, but these are referenced here as stimulus components.) Each stimulus component is associated in an all-or-none fashion with a particular action. When a stimulus is presented, a random sample is drawn from a corresponding subset of stimulus components. The proportion of stimulus components in a stimulus' subset associated with an action determines the probability of that action's being chosen in response to the stimulus. As a consequence of a conditioning rule, the stimulus components that are sampled may have their associations with the action changed.

In SST two stimuli are said to be similar to the extent that they have common stimulus components in their respective subsets. Hence, generalization in SST works as follows. Training with one stimulus will result in a high proportion of the stimulus components in its subset becoming associated with a particular action. If a second similar stimulus is then tested, the probability of its eliciting this same action will be higher because the components it has in common with the first stimulus will have become conditioned to the action. The strength of generalization is proportional to the size of the intersection and the strength with which the first stimulus was conditioned. When all the subsets of stimulus components of all stimuli are disjoint, then no generalization can occur. In this special case, the SST approach becomes an independent-associations approach.

Although SST's approach to similarity works out fairly well for generalization (La Berge, 1961; Carterette, 1961; Atkinson and Estes, 1963), it has difficulty with discrimination learning. With the SST approach, a learning system can never perform different actions to two similar stimuli with 100% reliability. Several directions have been taken in SST to handle this problem. One is based on the idea of somehow eliminating the effectiveness of stimulus components in the intersection of stimulus subsets when discrimination between the corresponding stimuli becomes necessary. Work along this line is usually based on the idea of selective attention (Restle, 1955; Zeaman and House, 1963; Lovejoy, 1968; Sutherland and Mackintosh, 1971). The other major direction taken to handle stimulus discrimination within SST is based on the idea that configurations of stimulus components may be perceived as whole patterns or "gestalts" (Atkinson and Estes, 1963; Friedman, Trabasso and Mosberg, 1967). Both of these remedies involve bringing in additional mechanisms to handle the problem of discrimination learning.

The Linear-Mapping Approach — A third approach to stimulus similarity has been widely used in pattern classification, control theory, neural network research, and animal learning theory. In the *linear-mapping approach*, stimuli are represented as vectors, where each component of the vector indicates the presence or absence (or extent) of a stimulus component. A linear mapping is formed from the stimulus vectors to a quantity that determines, usually via a threshold, the action to be taken. As a consequence, similarity and generalization between stimulus vectors is determined by the degree of their linear dependence, i.e., by their inner product.

When all stimulus vectors are mutually orthogonal, no stimuli are similar, and no generalization can occur. In this case, the linear-mapping approach reduces to a rather roundabout form of the independent-associations approach. The linear-mapping approach is the approach we studied in the research reported here. The tasks described in this section are concerned exclusively with binary-action problems in which the action $y[t]$ is either 1 or 0. Actions are selected as follows:

$$y[t] = \begin{cases} 1, & \text{if } s[t] + a[t] > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

where

$$s[t] = \sum_{i=1}^{n} x_i[t] w_i[t], \tag{6}$$

$\vec{x}[t] = (x_1[t], x_2[t], ..., x_n[t])$ is the stimulus vector presented at time t , $\vec{w}[t] = (w_1[t], w_2[t], \ldots, w_n[t])$ is a weight vector at time t, n is the total number of stimulus components, and $a[t]$ is a normally distributed random variable of mean 0 and standard deviation σ^y.

The linear-mapping approach is better at discrimination than is SST. In the SST approach, perfect discrimination between two stimuli is possible only if the stimuli are totally dissimilar. In the linear-mapping approach, two vectors can be very similar, with a great deal of generalization occurring between them, but complete discrimination can still be possible. Roughly, two stimulus vectors are similar in the linear-mapping approach according to the angle between them. As long as the angle between two vectors is not zero, it is possible to fit a hyperplane between them and therefore to discriminate between them. Therefore, very similar vectors can be mapped to completely different actions with as high a degree of reliability as desired.

Although the linear-mapping approach is in some respects a better approach to associative learning than either SST or the independent-associations approach, it obviously has limitations. Considered by itself, for example, the linear-mapping approach does not address the important problem of re-defining the components of stimulus vectors (see Section 2). However, for the moment we are satisfied with the linear-mapping approach despite this limitation because the problem of forming even linear maps remains largely unstudied for the problems in which we are interested, i.e., those involving associative learning under reinforcement feedback. One purpose of the experiments presented in this section is to show that the few algorithms that have been proposed for associative reinforcement learning using the linear-mapping approach fail when required to form linear maps in interaction with certain types of environments.

Tasks

In the experiments described in this section, 11 learning algorithms were applied to 12 learning tasks. As in Section 3, for each task and algorithm simulation runs were made for many values of a learning constant parameter α. For Tasks 1–10, the α values used were the powers of two from 2^1 to 2^{-11}. For Tasks 11 and 12, the α values used were the powers of two from 2^1 to 2^{-15}. For each task, algorithm, and α value, 100 simulation runs were made, each differing only in the initial seed of the pseudo-random number generator.

In this subsection we provide a complete description of the 12 tasks. Discussion of the rationale for using these particular tasks is deferred to the "Discussion" subsection. Table 3 summarizes the 12 tasks. Each facet of this table is fully discussed below.

As in Section 3, all tasks are binary-action tasks ($y[t] = 1$ or $y[t] = 0$). On each time step t the environment sends to the learning system one of two stimulus patterns, which we represent as vectors and denote $\vec{x}^1$ and $\vec{x}^2$. For Tasks 1–8,

$$\vec{x}^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{x}^2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}. \tag{7}$$

For Tasks 9–12,

$$\vec{x}^1 = \begin{pmatrix} .5 \\ .5 \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{x}^2 = \begin{pmatrix} 0 \\ 1.5 \\ 1.5 \end{pmatrix}. \tag{8}$$

In both cases the two stimuli are clearly distinguishable via the first and third components, but are also similar via the second component.

One difference between the stimuli of Tasks 1–8 and those of Tasks 9–12 is their intensity. On Tasks 1–8 the two stimuli are equally intense, whereas on Tasks 9–12 the first stimulus is half as intense as the stimuli on the other tasks, and the second stimulus is

Table 3

Associative Learning Tasks.

Task	Expected Value of Payoff				Stimulus Frequency		Stimulus Intensity			
	Stimulus $\vec{x}^1$		Stimulus $\vec{x}^2$							
Number	$y = 1$	$y = 0$	$y = 1$	$y = 0$	$\vec{x}^1$	$\vec{x}^2$	$\vec{x}^1$	$\vec{x}^2$	Steps	Description
1	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.5	.5	1	1	1500	Symmetrical
2	.1	−.1	−.1	.1	.5	.5	1	1	200	
3	−.6(.2)	−.8(.1)	.6(.8)	.8(.9)	.5	.5	1	1	1500	Payoff Level Asymmetry
4	−.6	−.8	.6	.8	.5	.5	1	1	200	
5	.05(.525)	−.05(.475)	−.15(.425)	.15(.575)	.5	.5	1	1	1500	Payoff Spread Asymmetry
6	.05	−.05	−.15	.15	.5	.5	1	1	200	
7	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.25	.75	1	1	1500	Stimulus Frequency Asymmetry
8	.1	−.1	−.1	.1	.25	.75	1	1	200	
9	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.5	.5	.5	1.5	1500	Stimulus Intensity Asymmetry
10	.1	−.1	−.1	.1	.5	.5	.5	1.5	200	
11	−.4(.3)	−.8(.1)	.3(.65)	.9(.95)	.25	.75	.5	1.5	3000	Combined Asymmetries
12	−.65	−.75	.55	.85	.25	.75	.5	1.5	2000	

one-and-a-half times as intense. This difference is indicated in Table 3 in the columns labeled "Stimulus Intensity."

On all tasks the stimulus presented is selected probabilistically. On Tasks 1–6 and 9–10, the two stimuli are presented with equal frequency, i.e., each with a probability of .5. On any given time step of Tasks 7–8 and 11–12, $\vec{x}^2$ is presented with a probability of .75 and $\vec{x}^1$ with a probability of .25. These probabilities are listed in Table 3 in the columns labeled "Stimulus Frequency."

Half of the tasks are binary-reinforcement tasks and half are continuous-reinforcement tasks. In the binary-reinforcement tasks, the reinforcement r received from the environment is either a $+1$ (success) or a -1 (failure), whereas in the continuous-reinforcement tasks the reinforcement can take on any real value. The odd-numbered tasks are binary-reinforcement tasks and the even-numbered tasks are continuous-reinforcement tasks.

In all tasks, the reinforcement $r[t+1]$ is a stochastic function of the preceding action $y[t]$ and stimulus $\vec{x}[t]$. Since there are two possible stimuli and two possible actions, there are four possible combinations of stimulus and action. Columns 3–6 of Table 3 list the expected value of the reinforcement for each task for each of these 4 combinations.

For the continuous-reinforcement tasks, the reinforcement is computed from the expected values in Table 3 as

$$r[t+1] = r_{y[t]}^{\vec{x}[t]} + a[t].$$

where $r_y^{\vec{x}}$ denotes the expected value of the reinforcement corresponding to stimulus $\vec{x}$ and action y, and where $a[t]$ is a normally distributed random variable of mean 0 and standard deviation σ^r. For all tasks except Task 12, $\sigma^r = .1$. For Task 12, $\sigma^r = .025$.

For the binary-reinforcement tasks, reinforcement is computed from the probabilities given in parentheses in Columns 3–6 of Table 3. Letting $P_y^{\vec{x}}$ denote $P\{r(\vec{x}, y) = 1\}$, i.e., the probability of a successful outcome given stimulus $\vec{x}$ and action y, the expected values given in Table 3 are computed as follows:

$$r_y^{\vec{x}} = 1 \cdot P_y^{\vec{x}} - 1 \cdot (1 - P_y^{\vec{x}}).$$

Finally, the last column of Table 3 provides a verbal description of the type of problem each pair of tasks presents to a learning algorithm. These descriptions are elaborated later in this section.

Algorithms

All 11 algorithms used in these experiments use the linear mapping approach to associative learning. That is, they all update a weight vector $\vec{w}[t] = (w_1[t], w_2[t], \ldots, w_n[t])$ and select their actions according to (5) and (6) with $\sigma^y = .3$ for all algorithms. The update rules of some algorithms use $\pi[t]$, the probability that $y[t] = 1$ given $\vec{x}[t]$. Determining $y[t]$ according to (5) implies that

$$\pi[t] = \Phi(s[t]/\sigma^y),$$

where Φ is the unit normal distribution function (an S-shaped function). All normally distributed random numbers used in these experiments were approximated by linear interpolation from tables and uniformly distributed random numbers.

The learning algorithms studied in this section's experiments differ only in the rules they use to update the weight vector $\vec{w}[t]$. The update rules and other relevant equations for the 11 algorithms used in these experiments are given in Table 4. The update rules are all simple extensions to the case of associative learning of the update rules of the algorithms discussed in Section 3.

Algorithms 4–7 are the most straightforward extensions of the nonassociative learning algorithms of Section 3 to associative learning. The update rules of Algorithms 4–7 of this section are very similar to those of Algorithms 4–7 of Section 3 (compare Tables 2 and 4). The primary difference (besides the fact that the new rules update all the components $w_i[t]$ of a vector $\vec{w}[t]$ whereas the rules of Section 3 update a scalar $w[t]$) is that the new update rules each have one more factor than the corresponding update rules of Section 3. In all cases the additional factor determining the modification of $w_i[t]$ is the corresponding stimulus vector component $x_i[t]$.

The update rule of each algorithm therefore includes a product of two factors, one depending on the action selected, either $y[t] - \frac{1}{2}$, $y[t] - \pi[t]$, $1 - y[t] - \frac{1}{2}$, or $1 - y[t] - \pi[t]$, and one depending on the presence or absence of a stimulus component, $x_i[t]$. Using Klopf's (1972) terminology, we call this product, denoted $e_i[t]$, the *eligibility* of w_i at

Table 4

Associative Learning Algorithms.

Number	Update Rule	Relevant Tasks
1	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \\ 0, & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
2	$w_i[t+1] = w_i[t] + \begin{cases} 0, & \text{if r[t+1]=-1} \\ \alpha(1 - y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \end{cases}$	1,3,5,7,9,11
3	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \\ \alpha(1 - y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \end{cases}$	1,3,5,7,9,11
4	$w_i[t+1] = w_i[t] + \alpha r[t+1](y[t] - 1/2)x_i[t]$	1-12
5	$w_i[t+1] = w_i[t] + \alpha r[t+1](y[t] - \pi[t])x_i[t]$	1-12
6	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - r[t])(y[t] - 1/2)x_i[t]$	3,4
7	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - r[t])(y[t] - \pi[t])x_i[t]$	3,4
8	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])(y[t] - 1/2)x_i[t]$	1-12
9	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])(y[t] - \pi[t])x_i[t]$	1-12
10	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - 1/2)x_i[t], & \text{if r[t+1]=1} \\ 0, & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
11	$w_i[t+1] = w_i[t] + \begin{cases} 0, & \text{if r[t+1]=-1} \\ \alpha(1 - y[t] - 1/2)x_i[t], & \text{if r[t+1]=1} \end{cases}$	1,3,5,7,9,11

Where:

$$w_i[0] = 0 \qquad v_i[0] = 0 \qquad y[t] \in \{1, 0\} \qquad \alpha > 0$$

$\pi[t]$ is the probability that $y[t] = 1$, given $\vec{x}[t]$

$$y[t] = \begin{cases} 1, & \text{if } s[t] + a[t] > 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $a[t]$ is a normally distributed random variable of mean 0 and standard deviation .3, and

$$s[t] = \sum_{i=1}^{n} w_i[t]x_i[t]$$

$$p[t] = \sum_{i=1}^{n} v_i[t]x_i[t] \qquad p[0] = r[1]$$

time t. It indicates the extent to which w_i is eligible for undergoing modification due to the reinforcement received at that time.

Like Algorithms 4–7, Algorithms 1–3 of this section also correspond to the like-numbered algorithms of Section 3 These algorithms, however, are somewhat less similar to their Section 3 counterparts then are Algorithms 4–7. Algorithms 1–3 of this section determine changes in $w[t]$ whereas the corresponding Section 3 algorithms directly determine changes in $\pi[t]$.

Algorithms 10 and 11 do not correspond to any algorithms discussed in Section 3. In this and the previous section we compare pairs of algorithms that differ only in that one uses $y[t] - \frac{1}{2}$ and the other uses $y[t] - \pi[t]$ in their update rules. Given our modification of Algorithms 1–3, all of which use $y[t] - \pi[t]$, it is possible also to include the corresponding algorithms that use $y[t] - \frac{1}{2}$. Algorithms 10 and 11 correspond to Algorithms 1 and 2 in this way. There was no need to add an algorithm for the $y[t] - \frac{1}{2}$ case corresponding to Algorithm 3 because Algorithm 4 already fills this role.

Reinforcement Comparison in Associative Learning — Algorithms 6 and 7 are simple reinforcement-comparison algorithms very similar to Algorithms 6 and 7 of Section 3. These algorithms compare the current reinforcement with the immediately preceding reinforcement. Although this technique worked well on the problems of Section 3, there are reasons to doubt their success on associative learning problems. It is much more difficult for an algorithm to compare current with past reinforcement levels properly in associative learning problems than it is in nonassociative learning problems. The problem is that for some tasks it may be possible to obtain high reinforcement levels in the presence of one stimulus but only low reinforcement levels in the presence of another. Tasks 3 and 4, for example, are of this sort. On such tasks, if different stimuli occur on two successive steps, then the change in reinforcement may be primarily due to the change in stimulus rather than to the action selected by the learning algorithm. In such cases the use of $r[t+1] - r[t]$ (as in Algorithms 6 and 7) is inappropriate. To illustrate this problem while minimizing the complexity of the experiments, Algorithms 6 and 7 were applied only to Tasks 3 and 4.

Whereas comparing the current reinforcement level with past reinforcement levels is a very simple process in nonassociative learning, in associative learning we see that it is much less straightforward. To obtain the maximum advantage from the reinforcement-comparison technique, the reinforcement received on the current step must be compared with the reinforcement received on previous steps ***on which the same stimulus occurred as occurred on the current step.*** How are the past reinforcement levels for each different stimulus to be recorded and kept separate from each other? The problem is particularly difficult if one assumes, as is done here, that the stimuli are not individually recognizable and separable, as they are in the independent-associations approach.

One way of solving this problem is to view the sequence of stimuli, and the reinforcement levels obtained by acting in response to them, as a training sequence for a supervised learning pattern classification algorithm. We used a version of the Widrow-Hoff, or Adaline, algorithm (Widrow and Hoff, 1960): A modifiable parameter vector $\vec{v}[t] = (v_1[t], v_2[t], ..., v_n[t])$ is required in addition to $\vec{w}[t]$. Whereas $\vec{w}[t]$ maps a stimulus to an action, $\vec{v}[t]$ maps a stimulus to an average of the reinforcement levels obtained when the stimulus (or similar stimuli) occurred in the past. We call $\vec{v}$ the ***reinforcement-association vector*** (whereas we call $\vec{w}$ the ***action-association vector***), and we call this average of past reinforcement levels the ***predicted reinforcement*** and denote it as $p[t] \in \Re$. It is computed from the current stimulus $\vec{x}[t]$ in the usual way for a linear-mapping approach to association:

$$p[t] = \sum_{i=1}^{n} v_i[t]x_i[t], \quad \forall\, t > 0.$$

We update $\vec{v}[t]$ so that $p[t]$ becomes an average of the appropriate past reinforcement levels. This rule changes $\vec{v}[t]$ according to the discrepancy between the predicted reinforcement $p[t]$ and the corresponding actual reinforcement $r[t+1]$:

$$v_i[t+1] = v_i[t] + \beta(r[t+1] - p[t])x_i[t], \tag{9}$$

for $t = 0, 1, \ldots,$ and $i = 1, \ldots, n$, where $v_i[0] = 0$, $p[0] = r[1]$, and $\beta = .1$ for all simulations discussed in this section. The discrepancy between expected reinforcement $p[t]$ and the actual reinforcement $r[t+1]$, in addition to being the error in the predicted reinforcement, is also the quantity needed as a comparison of the current reinforcement level with past reinforcement levels associated with the current stimulus. Algorithms 8 and 9 use this discrepancy in this way. For example, Algorithm 8 updates its weight vector $\vec{w}[t]$ as follows:

$$w_i[t+1] = w_i[t] + \alpha\,(r[t+1] - p[t])(y[t] - 1/2)x_i[t], \tag{10}$$

for $t = 0, 1, \ldots,$ and $i = 1, \ldots, n$. When the actual reinforcement $r[t+1]$ is greater (less) than the predicted reinforcement $p[t]$, w_i is changed so as to make the action selected more (less) likely as a response to the stimulus. Note that if the same stimulus were presented on every step, then the problem would no longer be an associative learning problem, and this reinforcement-comparison technique would work like that used in Section 3. In fact, if the single stimulus were a vector of a single component of value is 1, then (9) and (10) reduce exactly to the equations specifying Algorithm 8 of Section 3.

Although a reinforcement-comparsion algorithm for associative learning is necessarily more complex than those considered so far, such an algorithm could potentially perform much better than non-reinforcement-comparison algorithms. The results reported in Section 3 suggest that reinforcement-comparison algorithms learn much more rapidly than non-reinforcement-comparison algorithms on nonassociative learning tasks. It is likely, therefore, that reinforcement-comparison algorithms might have a similar advantage in associative-learning tasks. The results of the experiments of this section bear out this hypothesis.

Results

For each task, algorithm, and value of α, 100 simulation runs were made, each differing only in the initial seed for the random number generator. At the end of each run the final parameter vector $\vec{w}[101]$ was recorded. From this, the probability of the learning system selecting Action 1 on the next step, had the run been continued, was computed as follows:

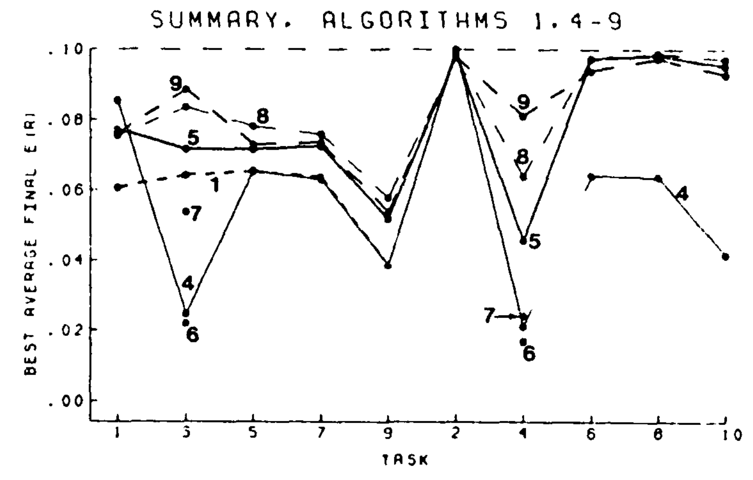
$$\pi^1 = P\{y[101] = 1 \mid x[101] = \vec{x}^1\} = \Phi(\sum_{i=1}^{3} w_i[101]x_i^1/\sigma^y)$$
$$\pi^2 = P\{y[101] = 1 \mid x[101] = \vec{x}^2\} = \Phi(\sum_{i=1}^{3} w_i[101]x_i^2/\sigma^y),$$

where $P\{\cdot \mid \cdot\}$ denotes a conditional probability. From these, the known stimulus presentation frequencies, and the expected values of the reinforcement as a function of stimulus and action (listed in Table 3), one can compute the expected value of the reinforcement on the next step, had the run continued:

$$\begin{aligned} E\{r[101]\} &= p^1\pi^1 r_1^1 + p^1(1-\pi^1)r_0^1 \\ &\quad + p^2\pi^2 r_1^2 + p^2(1-\pi^2)r_0^2, \end{aligned}$$

where p^i is the probability of stimulus $\vec{x}^i$ being presented, $i = 1, 2$, and r_j^i is the expected value of the reinforcement given that stimulus i and action j have occurred. These probabilities and expected values of the reinforcement depend on the task as listed in Table 3.

The expected value of the reinforcement on the next step after the end of the run is a good measure of how well the learning algorithm has learned by the end of the run. If it has learned rapidly and correctly, $\vec{w}$ will have reached a value for which the expected value of reinforcement is high. As with the data of Section 3, one good way to interpret these data is to compare the highest performance levels attained with each algorithm (each with its own best value for α). Henceforth, when we talk of the performance of an algorithm on a task, we will mean its *best* performance on that task, i.e., its performance with the α value that resulted in the highest performance level. Using this performance measure, Figure 20 summarizes the data giving the performances of all algorithms on the first 10 tasks.



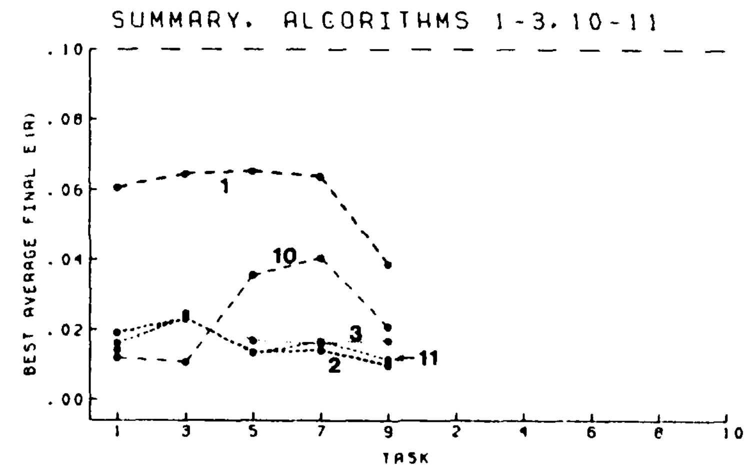


Figure 20

Summary of Algorithm Performance on Tasks 1–10 of Section 4.

Discussion

The 12 tasks were chosen to help us investigate the problems of stimulus discrimination in the face of asymmetrical treatment of the stimuli. For a particular task and stimulus, the preferable action is the one with the highest expected value of reinforcement, as given in Table 3. On all tasks, the correct action when stimulus $\vec{x}^1$ is presented differs from that when stimulus $\vec{x}^2$ is presented. To maximize reinforcement, a learning algorithm must discriminate between, and respond differently to, the two stimuli. For convenience, all tasks were constructed so that Action 1 is the best action when stimulus $\vec{x}^1$ is presented, and Action 0 is the best when stimulus $\vec{x}^2$ is presented.*

On all tasks the two stimulus vectors $\vec{x}^1$ and $\vec{x}^2$ are similar and yet distinguishable. For Tasks 1–8, these vectors are as given by (7), and for Tasks 9–12, they are as given by (8). The vectors $\vec{x}^1$ and $\vec{x}^2$ are similar by virtue of their second components, which are both positive, and they are distinguishable by virtue of their first and third components, which are positive for one stimulus and zero for the other.

When $\vec{x}^1$ occurs, the first and second weight vector components are modified. When $\vec{x}^2$ occurs, the second and third weight vector components are modified. Since in the first case Action 1 is correct, and in the second, Action 0 is correct, the first weight vector component w_1 becomes positive (associated with Action 1) and the third weight vector component w_3 becomes negative (associated with Action 0), but it is unclear what happens to the second weight vector component w_2. In the ideal case, the countervailing influences on w_2 would cancel out, leaving w_2 near zero. This result would be ideal because, by (5) and (6), it would leave the action in response to $\vec{x}^1$ determined by w_1 and the action in response to $\vec{x}^2$ determined by w_3. This result would most likely occur on tasks that are symmetrical with respect to the two stimuli. True symmetry guarantees that the countervailing influences on w_2 are of equal strength.

The tasks of this section were designed in pairs, 1 with 2, 3 with 4, etc., each pair consisting of one binary-reinforcement task and one continuous-reinforcement task. The first pair of tasks, Tasks 1 and 2, are completely symmetrical with respect to the two stimuli. The next 4 pairs, 3–4, 5–6, 7–8, and 9–10, treat the two stimuli asymmetrically in one, and only one, way. Finally, the tasks of the pair 11–12 combine all 4 forms of asymmetry.

The continuous-reinforcement tasks used in these experiments are significantly easier

* The learning algorithms do not "know" which stimulus had which number, so there is no way that they can take advantage of this uniformity.

than the binary-reinforcement tasks. For this reason, the experiments with the continuous-reinforcement tasks were run for fewer steps than those with the binary-reinforcement tasks (200 steps as opposed to 1500). If both types of experiments had been run for 1500 steps, performance differences between some algorithms on the easier tasks would have been lost in a ceiling effect since many algorithms would have performed near optimally. Tasks 11 and 12 were run much longer than the other tasks for reasons to be discussed below.

In the following discussions we occasionally refer to particular performance differences between algorithms as being either "significant" or "not significant." By this we are referring to statistical significance at the $P = .05$ probability criterion.

Tasks 1 and 2: No Asymmetries — Tasks 1 and 2 are completely symmetrical with respect to the two stimuli. On these tasks, the stimuli are presented with equal frequency and equal intensity. On these tasks, the expected values of reinforcement associated with the two stimuli are equal (although which action has the higher expected value switched from one stimulus to the other). As discussed above, this symmetry makes these tasks the *least* likely to be influenced by the problems of discrimination between similar stimuli.

Tasks 1 and 2 are closely related to Tasks 3 and 6 of Section 3. These tasks have similar distributions of expected value of reinforcement as a function of the action selected. On all four of these tasks the expected value of the reinforcement is positive when one action is chosen and negative, by the same amount, when the other action is chosen. For such tasks we say that the reinforcement is unbiased, i.e., its expected value as a function of action is distributed symmetrically around zero.

The similarity between Tasks 1 and 2 of this section and 3 and 6 of Section 3 is born out in similar performances of corresponding algorithms on corresponding tasks (compare Figure 1 and Figure 20). As described in Section 3, Algorithms 4–9 performed essentially perfectly on the continuous-reinforcement task (Task 6) whereas here the closely related Algorithms 4–9 also performed essentially perfectly on the corresponding continuous-reinforcement task (Task 2). The binary-reinforcement Task 3 of Section 3 corresponds to the binary-reinforcement Task 1 of the present section. The ranking of performance of the algorithms is very similar on these two tasks. On both tasks, Algorithm 4 performed best, followed by Algorithms 5, 8, and 9, whose performances were not significantly different from each other, followed by Algorithm 1, which was in turn distantly followed by Algorithms 2 and 3. The two new algorithms added in this section, Algorithms 10 and 11, fell in with the last group of poorest performing algorithms.

Tasks 3 and 4: Reinforcement-Level Asymmetry — On Tasks 3 and 4, when stimulus $\vec{x}^1$ occurs, the expected value of reinforcement is always high, and when stimulus $\vec{x}^2$ occurs, the expected value of reinforcement is always low. These tasks differ from Tasks 1 and 2 only in that the expected value of the reinforcement is biased in this way. We call this asymmetry in the treatment of stimuli *reinforcement-level asymmetry*. In Section 3 we discussed nonassociative tasks that differed from each other in a similar way. Ignoring for the moment the generalization between the two stimuli, Tasks 1 and 2 of this section are similar to Tasks 3 and 6 of Section 3, and Tasks 3 and 4 of this section are similar to a combination of Tasks 1 and 2, and 4 and 5, respectively, of Section 3.

In Section 3 we reported that Algorithm 4 performed the best of all algorithms on the unbiased tasks and poorly on the biased tasks, and we find a similar result in Tasks 1–4 of this section. The upper graph of Figure 20 shows that whereas Algorithm 4 performed best of all algorithms on Tasks 1 and 2, it performed very poorly on Tasks 3 and 4. Along these same lines, we observed in Section 3 that whereas Algorithm 5 was not a significantly worse performer than the reinforcement-comparison algorithms, such as Algorithms 8 and 9, on the unbiased tasks, it was a significantly worse performer for the biased tasks. For Tasks 1–4 of this section we see a similar result. Algorithms 5, 8, and 9 performed nearly identically on the unbiased tasks, Tasks 1 and 2, whereas 8 and 9 performed significantly better than 5 on the biased tasks, Tasks 3 and 4.

On Tasks 3 and 4, the expected value of the reinforcement varies more with the stimulus presented than with the action selected. This reinforcement-level asymmetry poses a particular challenge to reinforcement-comparison algorithms. Reinforcement-comparison algorithms that compare successive reinforcement levels without regard to the stimuli presented, such as Algorithms 6 and 7, had a difficult time with these tasks as shown in

Figure 20. Algorithms 6 and 7 clearly performed much worse than Algorithm 5, which is not a reinforcement-comparison algorithm, and much worse than Algorithms 8 and 9, which are more sophisticated reinforcement-comparison algorithms.

The performances of the binary-reinforcement Algorithms 1, 2, and 3 on Tasks 1 and 3 were similar to the perfomances of the corresponding algorithms of Section 3. Algorithms 2 and 3 performed very poorly on both tasks. Algorithm 1 achieved an intermediate performance level, significantly better than Algorithms 2 and 3, and significantly worse than Algorithms 5, 8, and 9.

Tasks 5 and 6: Reinforcement-Spread Asymmetry — In Tasks 5 and 6 the difference in the expected value of reinforcement for the two actions is much larger for one stimulus than it is for the other. For stimulus $\vec{x}^1$, the expected value of the reinforcement is .1 higher when the correct action is chosen than when the incorrect action is chosen. For stimulus $\vec{x}^2$, the expected value of the reinforcement changes by .3 (a large amount under the circumstances) depending on whether the correct or the incorrect action is selected. These tasks are asymmetrical in the *spread* between their reinforcement levels for correct and incorrect actions. In a task with reinforcement-spread asymmetry it is more important to learn which action is correct for one stimulus than it is for the other. Learning will generally be more rapid when the reinforcement spread is large than when it is small. Because of this, reinforcement-spread asymmetry tends to cause the correct action for one stimulus (in this case, stimulus $\vec{x}^2$) to be learned much more rapidly than that for the other. When the two stimuli are similar, this more rapid learning generalizes strongly to the other stimulus. When two different actions must be learned to the two stimuli, there is a danger that generalization from the more rapidly learning stimulus will overwhelm the learning to the other stimulus and cause the incorrect action to become associated with it.

One gets a sense of which algorithms are susceptible to this difficulty by comparing the relative performance rankings of the various algorithms on Tasks 5–6 and Tasks 1–2. It is not meaningful to compare the *absolute* performance of algorithms between these pairs of tasks. On the one hand, Tasks 5–6 might be expected to be more difficult than Tasks 1–2 because of reinforcement-spread asymmetry. On the other hand, Tasks 5–6 might be expected to be easier than Tasks 1–2 because, for one stimulus, the reinforcement-spread is larger than in Tasks 1 and 2, which should result in faster learning. Overall it is not possible to predict with certainty which pair of tasks is the more difficult to master.

The changes in the relative performance rankings of algorithms from Tasks 1–2 to Tasks 5–6 suggest the following conclusions. First, Algorithm 4 is strongly affected by reinforcement-spread asymmetry. Whereas in Tasks 1 and 2 it performed the best of all algorithms, in Tasks 5 and 6 it performed significantly worse than Algorithms 5, 8, and 9. Second, all the binary-reinforcement algorithms, Algorithms 1, 2, 3, 10, and 11, performed significantly worse than the better performing algorithms (5, 8, and 9). For some reason Algorithm 10 performed significantly better vis-a-vis Algorithms 2, 3, and 11 on Task 5 than it did on task 1, but its performance was still very poor.

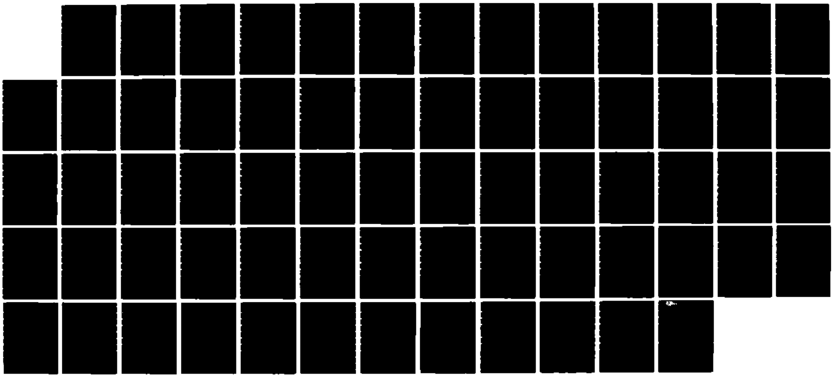
The effect of reinforcement-spread asymmetry on the performances of Algorithms 5, 8, and 9 is less clear. On Tasks 1 and 2 all three algorithms performed equally well. On Task 5 Algorithm 8 performed better than 5 and 9, but only the improvement over Algorithm 5 is statistically significant. On Task 6, Algorithms 5 and 8 both performed significantly better than Algorithm 9.

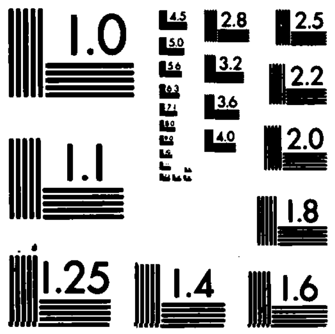
Tasks 7 and 8: Stimulus-Frequency Asymmetry — In Tasks 7 and 8 one stimulus is presented three times as frequently as the other. Learning about the more frequently presented stimulus generally occurs more rapidly than learning about the other, simply because there is more experience with it. This asymmetry in learning rate can lead to an overwhelming of the more slowly forming association by generalization from the faster one, as discussed above for reinforcement-spread asymmetry.

The effect of this *stimulus-frequency asymmetry* on the performances of the various algorithms was very similar to the effect of reinforcement-spread asymmetry. Algorithm 4 was strongly affected, performing significantly worse than Algorithms 5, 8, and 9, whereas it was the best performer on Tasks 1 and 2. On Task 7 the algorithms designed for binary-reinforcement tasks, Algorithms 1, 2, 3, 10, and 11, produced strikingly similar performances to those they produced on Task 5. Algorithm 1 was best, performing at the

AD-A140 295 SIMULATION EXPERIMENTS WITH GOAL-SEEKINN ADAPTIVE ELEMENTS(U) MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INNORMATION SCIENCE A G BARTO FEB 84 2/2

UNCLASSIFIED AFWAL-TR-84-1022 F33615-80-C-1088 F/G 6/4 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

same level as Algorithm 4. Algorithm 10 was again significantly better than 2, 3, and 11, though again, it still performed poorly.

The effect of stimulus-frequency asymmetry on Algorithms 5, 8, and 9 is unclear (as is the effect of reinforcement-spread asymmetry). Recall that on Tasks 1 and 2 these three algorithms performed equally well. On Task 7 their performances fell in the same rank order as they did on Task 5 (8 then 9 then 5), but here none of the differences are statistically significant. On Task 8, Algorithms 8 and 5 were better than Algorithm 9, but only Algorithm 8 significantly so.

Tasks 9 and 10: Stimulus-Intensity Asymmetry — In Tasks 9 and 10 one of the two stimuli is three times as intense as the other (the two stimuli are given in (8)), whereas in Tasks 1 and 2, the stimuli have the same intensity (see (7)). This is the only difference between Tasks 9 and 10 and Tasks 1 and 2.

This *stimulus-intensity asymmetry* also causes an action to be learned more rapidly for one stimulus than for the other. All learning algorithms considered in these experiments use a factor ($x_i[t]$) that depends on the intensity of the stimulus presented. Larger changes are thus made in the weight vector $\vec{w}$ on those trials on which the more intense stimulus vector occurs than on the trials on which the less intense stimulus vector occurs. As in stimulus-frequency asymmetry and reinforcement-spread asymmetry, this asymmetry in the rate of learning creates the danger of the more slowly forming association being overwhelmed by generalization from the faster one.

The results for Tasks 9 and 10 are similar to those on Tasks 3–8. Algorithm 4 was again strongly affected by the asymmetry, performing significantly worse than Algorithms 5, 8, and 9, whereas it was the best performer on Tasks 1 and 2. Algorithm 1 performed almost exactly the same as Algorithm 4. The other binary-reinforcement algorithms, Algorithms 2, 3, 10, and 11, had the poorest performance of all algorithms. Although Algorithms 5, 8, and 9 maintained the same ordering as they did on Tasks 7 and 8 (8 then 5 then 9), there were no significant differences between their performances.

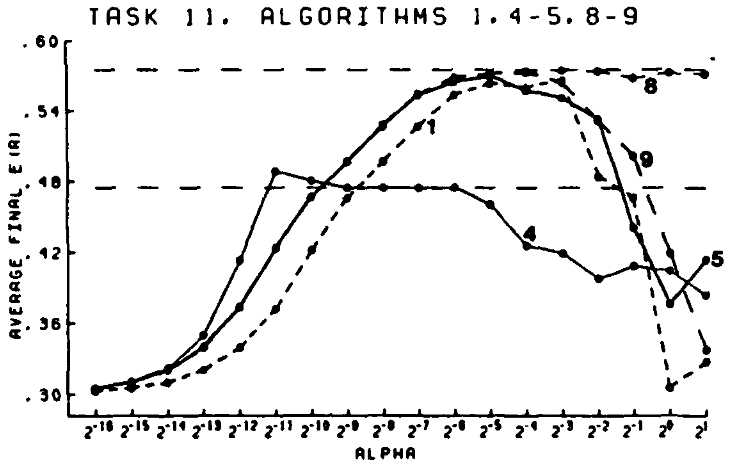
Tasks 11 and 12: Combined Asymmetries — Tasks 11 and 12 combine all four of the asymmetries present individually in Tasks 3–10. Reinforcement-level asymmetry, reinforcement-spread asymmetry, stimulus-frequency asymmetry, and stimulus-intensity asymmetry are combined in such a way that their effects are additive rather than subtractive. As discussed above, each of these asymmetries results in learning to one stimulus proceeding at a more rapid rate than learning to the other stimulus. In Tasks 11 and 12, the favored stimulus with respect to all three asymmetries is the same stimulus, $\vec{x}^2$. Stimulus $\vec{x}^2$ is presented with greater frequency, with greater intensity, and has a greater reinforcement-spread, than stimulus $\vec{x}^1$. The combination of these asymmetries makes the task much more difficult than does any of them alone.

The experiments involving Tasks 11 and 12 ran for many more steps (see Table 3) than the experiments involving the other tasks. One reason for this is that Tasks 11 and 12, since they combine four asymmetries, were expected to be more difficult than tasks 1–10. However, Tasks 11 and 12 contain weaker forms of these asymmetries than are present in Tasks 3–10. In addition, the standard deviation of the noise in the reinforcement for the continuous-reinforcement task (Task 12) is lower ($\sigma = .1$ instead of $\sigma = .025$), and the reinforcement spreads for the binary-reinforcement task (Task 11) are larger. These differences make both of these tasks less difficult than Tasks 3–10.

The real motivation behind the selection of Tasks 11 and 12 was to study the convergence behavior of the various algorithms. With these tasks we were less interested in how rapidly learning proceeds and more interested in the ability of various algorithms to ultimately choose the correct actions.

The results of Tasks 11 and 12 are not included in Figure 20, but are shown in Figures 21 and 22. The experiments appear to have been run long enough for all algorithms to show the performance levels to which they would have converged, with the exception of Algorithm 2 on Task 11 and Algorithm 5 on Task 12. All the other algorithms appear to converge to one of two performance levels, shown as two dashed lines on each graph.

The higher of the two performance levels indicated by dashed lines is the maximum possible performance level for these tasks. This performance level would have been ob-



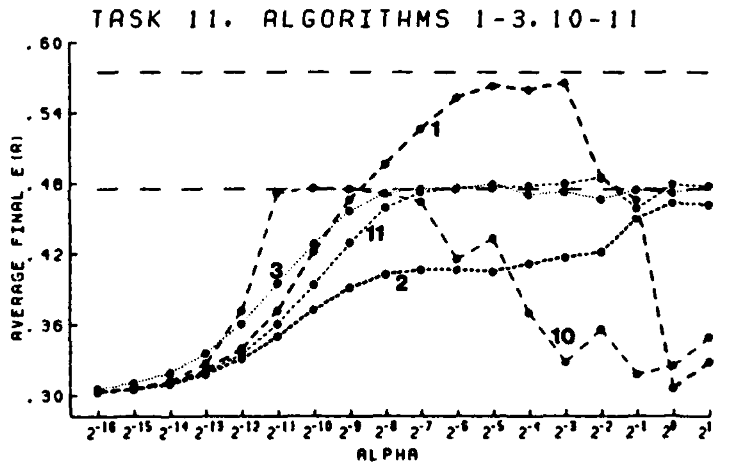


Figure 21

Algorithm Performance on Task 11 of Section 4.

tained were the optimal action chosen with probability one for both stimuli. Algorithms 1, 5, 8, and 9 appear to have converged to this performance level on Task 11, and Algorithms 8, and 9 appear to have converged to this performance level on Task 12. Task 12 appears not to have been run long enough for Algorithm 5 to converge. The highest performance level for Algorithm 5 is significantly lower than that of Algorithms 8 and 9. Most likely, Algorithm 5 also converges to the optimal performance level on this task but converges more slowly than Algorithms 8 and 9.

The lower of the two dashed lines shown in each graph of Figures 21 and 22 indicates the

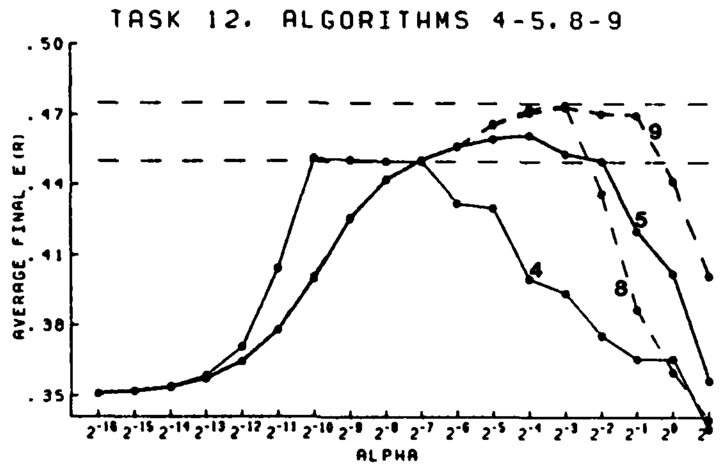


Figure 22

Algorithm Performance on Task 12 of Section 4.

expected performance level achieved if Action 0 is selected with probability one in response to both stimuli. This action is the better action in response to one stimulus but the poorer action in response to the other. This is the performance level that is obtained by a system that learns the correct action for the stimulus for which there is faster learning and allows this learning to overwhelm correct learning to the other stimulus through generalization. Convergence to this performance level indicates a failure to discriminate. The performances of Algorithms 3, 4, 10, and 11 appear to converge to this level on Task 11, and the performance of Algorithm 4 appears to converge to this level on Task 12.

These results on Tasks 11 and 12 do not *prove* that any algorithm converges or does not converge. Algorithms that appeared to converge to one of the performance levels indicated by dashed lines may actually converge to slightly different levels. Algorithms that failed to converge during the run of these experiments, or that appeared to converge to a suboptimal level, may converge to the optimal performance level if sufficiently small α values are used and the experiments are run long enough. The only way to definitively prove convergence of such algorithms is by mathematical analysis. These results do, however, strongly suggest what the results of such an analysis might be and provide practical indications of how the algorithms are likely to perform.

Conclusion

There are a number of similarities between the data of this section and those of Section 3. Here, as there, the algorithms designed purely for binary-reinforcement tasks (Algorithms 1, 2, 3, 10, and 11), with the exception of Algorithm 1, performed uniformly poorly across tasks. In the experiments of both sections, Algorithm 1 performed better than the other binary-reinforcement algorithms, but performed clearly worse than Algorithms 5, 8, and 9. In both sections, Algorithm 4 performed erratically across tasks, performing best on a few, very poorly on a few, and significantly worse than the best algorithms on the others.

As anticipated, the algorithms that simply compare the current and the immediately preceding reinforcement levels, Algorithms 6 and 7, performed much worse on these associative-learning tasks than they did on the nonassociative tasks of Section 3. These algorithms also performed significantly worse than Algorithms 5, 8, and 9 on the tasks of the present section.

The best performing algorithms across the tasks of this section were Algorithms 5, 8, and 9. It is not entirely clear which of these performed best. Of the three, Algorithm 9 performed significantly better on Tasks 3 and 4, but significantly worse than Algorithm 8 on Tasks 8 and 10, and significantly worse than both Algorithms 5 and 8 on Task 6. The only completely clear ordering that can be made among these three algorithms is that Algorithm 8 was better than Algorithm 5. Algorithm 8 performed as well or better than Algorithm 5 on every task, and significantly better on Tasks 3, 4, 5, and 10. In addition, both Algorithm 8 and Algorithm 9 performed significantly better than Algorithm 5 on Tasks 11 and 12.

Whether Algorithm 8 or Algorithm 9 is better apparently depends on the nature of the task. Although the performance of Algorithm 8 was better than or equal to that of Algorithm 9 on all tasks except Tasks 3 and 4, it is possible that "real-life" learning tasks resemble Tasks 3 and 4 more closely than any of the other tasks considered. Each of these tasks is a special case, and we have made no attempt to characterise what a real "typical case" might look like. However, the fact that on these tasks Algorithm 8 makes a serious bid for the place of best performer is significant in light of the results of Section 3. In the experiments described in Section 3, Algorithm 8 performed strictly inferior or equal to Algorithm 9. The change from nonassociative to associative learning tasks makes Algorithm 8 a better performer vis-a-vis Algorithm 9.

Since the performance of Algorithm 5 was strictly inferior to that of Algorithm 8 on these tasks, these results corroborate the results of Section 3 with regard to reinforcement-comparison algorithms. In the experiments of both sections the best performing algorithms across tasks were reinforcement-comparison algorithms.

As discussed earlier, and as is illustrated by the poor performance of the reinforcement-comparison algorithms, Algorithms 6 and 7, construction of a satisfactory reinforcement-comparison algorithm for associative learning is not a trivial task. The difficulties may be part of the reason researchers have avoided reinforcement-comparison algorithms, even for some types nonassociative learning. In this regard, the excellent performance of the more sophisticated reinforcement-comparison algorithms, Algorithms 8 and 9, is particularly important. This result demonstrates that algorithms can be designed for associative learning that can benefit from the advantages of a reinforcement-comparison mechanism. The advantages of reinforcement comparison can be obtained in associative-learning tasks as well as in nonassociative-learning tasks.

Perhaps the most important result of the experiments described in this section is the poor performance of Algorithm 4. Not only did Algorithm 4 perform much worse than Algorithms 5, 8, and 9 on all tasks involving any form of stimulus asymmetry, but the results of Tasks 11 and 12 suggest that this algorithm is unable to discriminate properly. On Tasks 11 and 12 this algorithm appears to converge to the incorrect choice of action. Yet Algorithm 4 appears to be the most straightforward implementation of the basic principle of associative reinforcement learning using the linear-mapping approach. In fact, of the algorithms considered here, Algorithm 4 is the algorithm most nearly like that of Farley and Clark (1954), which is one of the few associative reinforcement learning algorithms using the linear-mapping approach that has been computationally investigated. This also appears to be essentially the algorithm informally discussed by Minsky and

Selfridge (1961). The failure of this common-sense rule to produce effective learning in all but the simplest situations suggests that associative reinforcement learning involves subleties that were not recognised by early AI and cybernetic researchers.

Section 5
DELAYED REINFORCEMENT

Introduction

The simulations discussed in previous sections concerned learning problems in which actions affect only the reinforcement received on the next time step. We call such tasks *immediate-reinforcement* tasks. Tasks in which the effect of an action on reinforcement is delayed by two or more time steps we call *delayed-reinforcement* tasks. Delayed reinforcement can create a difficult (temporal) credit-assignment problem. When the reinforcement due to an action immediately follows that action, the only difficulty in assigning credit to the action is in interpreting the reinforcement as being higher or lower than usual (the reinforcement-comparison problem).* If there is additional uncertainty about which of the preceding actions caused the reinforcement, credit assignment becomes more difficult. The *uncertainty* that usually accompanies delayed reinforcement, rather than the delay itself, causes the difficulty. If reinforcing events are always delayed by a fixed amount known *a priori*, then the design of the learning system can take into account the fixed delay, and credit assignment need not be more difficult than it is with immediate reinforcement. On the other hand, if the delay is not known, then it increases uncertainty about the causal relationship between action and reinforcement, which makes temporal credit assignment more difficult. One cannot expect to find algorithms that completely eliminate this difficulty. One can, however, attempt to find learning algorithms whose performance degrades gracefully as uncertainty due to delay between action and reinforcement increases. As we show below, the algorithms we have considered so far require modification before they can work effectively on delayed reinforcement tasks in which no *a priori* knowledge of the delay is assumed.

Issues concerning reinforcement anticipation and secondary reinforcement were excluded from the experiments discussed in this section. In none of the tasks considered do

* Although there will remain the difficulty of assigning credit to the individual decisions determining the action (structural credit assignment).

actions affect subsequent neutral stimuli. All tasks considered in the present section either have no neutral stimuli or have neutral stimuli selected randomly. Therefore, there is no way for a learning system to use neutral stimuli to anticipate reinforcement, and thus no opportunity for secondary reinforcement to contribute to the learning process.

Eligibility Traces

How can the algorithms described in the preceding two sections be generalized to work with delayed as well as immediate reinforcement? We set aside for the moment the problem of generalizing reinforcement-comparison mechanisms. Assume that we have available a reinforcement signal $\hat{r}$ that already incorporates reinforcement comparison, but whose evaluations are delayed.

Perhaps the most general heuristics for assigning credit in the face of delayed reinforcement are those of *frequency* and *recency*. According to the *frequency heuristic*, one assigns credit to past decisions according to how many times they occurred. If one action had been made once in response to a particular stimulus in the time preceding reinforcement, and another action had been made twice, then the second action, according to the frequency heuristic, is twice as likely to have caused the reinforcement and thus deserves twice as much credit for it. According to the *recency heuristic*, one assigns credit for current reinforcement to past actions according to how recently they were made. Perhaps the simplest way of implementing this heuristic is to assign credit according to a monotonically decreasing function of the time between action and reinforcement that approaches zero as this time approaches infinity. One way of doing this is to assign credit according to an exponentially decreasing function λ^k of number k of time steps elapsing between action and reinforcement.

The following learning rule combines the frequency heuristic and an exponentially decaying recency heuristic:

$$w_i[t+1] = w_i[t] + \alpha\,\hat{r}[t+1](1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_i[t-k], \qquad (11)$$

for $t > 0$ and $i = 1, \ldots, n$ where $\vec{w}[t] = (w_1[t], \ldots, w_n[t])$ is the vector of action-association weights (mapping stimuli to actions), $e_1[t]$ is the eligibility of its ith component, as discussed in Section 4, α is a positive learning constant, and $\hat{r}[t+1]$ is the heuristic reinforcement. The eligibility $e_i[t]$ can be regarded as the credit that should be assigned to w_i for a "unit credit" assigned to the behavior at time t. λ^k is the credit that should be assigned, according to an exponential recency heuristic, to the behavior at time $t - k$ given that a unit reinforcement was received at time $t + 1$. The product $\hat{r}[t+1]\lambda^k e_i[t-k]$ therefore is the credit due to the behavior at $t-k$ that should be assigned to w_i given $\hat{r}[t+1]$. The frequency heuristic is implemented in this rule by summing up the credit due to the behavior at all past times. The factor $(1 - \lambda)$ in the learning rule normalizes the sum. If λ is near 1, then the credit assigned to an action falls off slowly as a function of increasing time between that action and the occurrence of a reinforcing event; if λ is near 0, then credit falls off rapidly. For $\lambda = 0$, (11) reduces to the form used in the learning rules of Section 4. In describing the experiments to follow, we use the parameter $\delta = 1 - \lambda$.

Equation 1 implements frequency and recency heuristics by means of an exponentially decaying backwards-averaging kernel. It is reasonable to regard a kernel of this general form as appropriate for learning situations in which nothing specific is known about underlying cause and effect relationships. Alternatively, it might be justifiable to assign credit according to an inverted U-shaped function of the time between action and reinforcement as suggested by Klopf (1972, 1982). This choice could be regarded as reflecting the distribution of the durations of the feedback pathways in which the learning system is embedded. In general, one would expect learning performance to improve to the extent that the shape of this kernel incorporates knowledge about the expected temporal relationship between cause and effect for particular types of transactions with particular environments. Ideally, perhaps, the kernel should resemble the cross-correlation of the action and reinforcement processes. In our research to date, no attempt has been made to use any knowledge of this type even though *some* amount of such knowledge is likely to be available about specific environments. We have also not considered algorithms for adaptively adjusting the form of kernel (e.g., by adjusting λ in the kernel above or by estimating the action/reinforcement cross-correlation), but such methods would have obvious utility.

A major advantage of the exponential decay form of the recency heuristic is that it allows the sum in (11) to be computed iteratively. Let us define a *trace operator* that maps any time sequence e_i into the time sequence $\bar{e}_i$ defined by

$$\bar{e}_i[t] = (1-\lambda)\sum_{k=0}^{t-1} \lambda^k e_i[t-k]$$

for $t > 0$, where $\bar{e}_i[0] = 0$. Then the algorithm for iteratively computing this trace can be derived as follows:

$$\begin{aligned} \bar{e}_i[t] &= (1-\lambda)\sum_{k=0}^{t-1} \lambda^k e_i[t-k] \\ &= (1-\lambda)e_i[t] + (1-\lambda)\sum_{k=0}^{t-2} \lambda^{k+1} e_i[t-1-k] \\ &= (1-\lambda)e_i[t] + \lambda \bar{e}_i[t-1], \end{aligned} \tag{12}$$

for $t > 0$. Although it requires only a single memory variable per component of $\vec{w}$, the above iterative algorithm is equivalent to remembering all past behavior and then applying frequency and exponential recency heuristics as in (11). The algorithm given by (12) is a standard recursive linear discrete-time filter.

When e_i is an eligibility time sequence as defined in Section 4, we call $\bar{e}_i$ an *eligibility trace.* All algorithms of this chapter update $\vec{w}$ by (11), which we rewrite using $\bar{e}_i$ as:

$$w_i[t+1] = w_i[t] + \alpha\, \hat{r}[t+1]\bar{e}_i[t]. \tag{13}$$

When $\lambda = 0$ ($\delta = 1$), $\bar{e}_i$ reduces to e_i, and (13) reduces to the same form as used in the algorithms of Section 4. For the even numbered algorithms of this section (4, 8, and 10) $e_i[t]$ is defined as $(y[t] - \frac{1}{2})x_i[t]$, and for the odd-numbered algorithms (5, 9, and 11) $e_i[t]$ is defined as $(y[t] - \pi[t])x_i[t]$, where $y[t] \in \{0, 1\}$ is the action taken at time t, $\pi[t]$ is the probability that $y[t] = 1$, and $x_i[t]$ is the ith component of the stimulus vector at time t. Otherwise, the algorithms differ only in the reinforcement-comparison mechanism used in forming $\hat{r}$, a subject we consider next. Table 5 summarizes all of this section's algorithms.

Reinforcement Comparison Under Delayed Reinforcement

One can view reinforcement-comparison algorithms as constructing a reinforcement signal $\hat{r}$ by comparing the primary reinforcement r with a *predicted reinforcement* $p[t]$:

$$\hat{r}[t+1] = r[t+1] - p[t]. \tag{14}$$

We call any reinforcement signal, such as this one, that is computed by the learning system a *heuristic reinforcement* signal. As discussed in Section 4, the predicted reinforcement $p[t]$ is computed from the stimulus vector $\vec{x}[t]$ by means of a *reinforcement-association vector* $\vec{v}[t]$:

$$p[t] = \sum_{i=1}^{n} v_i[t]x_i[t]. \tag{15}$$

The reinforcement association vector $\vec{v}$ is updated according to a variation of the Widrow-Hoff rule (Widrow and Hoff, 1960):

$$v_i[t+1] = v_i[t] + \beta\, \hat{r}[t+1]x_i[t], \tag{16}$$

where $v_i[0] = 0$, and β is a positive constant. Since $p[t]$ is an estimate of $r[t+1]$, $\hat{r}[t+1] = r[t+1] - p[t]$ is an error term. This learning rule correlates these errors with the stimuli that were present immediately before them and adjusts $\vec{v}$ in such a way as to reduce the error.

One way to generalize (16) to delayed reinforcement is to correlate the error $\hat{r}$ with all preceding stimuli, weighted according to their frequency and their recency in a manner similar to that discussed above for weighting past eligibilities:

Table 5

Learning Algorithms of Section 5

Number	Update Rule
4,5	$w_i[t+1] = w_i[t] + \alpha r[t+1]\bar{e}_i[t]$
8,9	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] - p[t])\bar{x}_i[t]$
10,11	$w_i[t+1] = w_i[t] + \alpha(r[t+1] + p[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] + p[t])\bar{x}_i[t]$

Where:

$$e_i[t] = \begin{cases} (y[t] - 1/2)x_i[t], & \text{for even numbered algorithms;} \\ (y[t] - \pi[t])x_i[t], & \text{for odd numbered algorithms.} \end{cases}$$

$$w_i[0] = 0 \qquad v_i[0] = 0 \qquad y[t] \in \{1, 0\} \qquad \alpha > 0 \qquad \beta = .05 \qquad 0 < \gamma < 1$$

$$y[t] = \begin{cases} 1, & \text{if } s[t] + \eta[t] > 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation .1, and

$$s[t] = \sum_{i=1}^{n} w_i[t]x_i[t]$$

$\pi[t]$ is the probability that y[t]=1, given $\bar{x}[t]$

$$p[t] = \sum_{i=1}^{n} v_i[t]x_i[t] \qquad p[0] = r[1]$$

For any time sequence $z[t]$, $\bar{z}[t]$ is defined by

$$\bar{z}[t] = (1-\delta)\bar{z}[t-1] + \delta z[t] \qquad \bar{z}[0] = 0 \qquad 0 < \delta < 1$$

$$\begin{aligned} v_i[t+1] &= v_i[t] + \beta\, \hat{r}[t+1](1-\lambda)\sum_{k=0}^{t-1} \lambda^k x_i[t-k] \\ &= v_i[t] + \beta\, \hat{r}[t+1]\bar{x}_i[t+1], \end{aligned} \tag{17}$$

where λ, $0 < \lambda < 1$, is an exponential decay rate as discussed above.

Equations (14), (15) and (17) constitute the mechanism used by Algorithms 8 and 9 of this section for constructing and updating the heuristic reinforcement signal. Algorithms 4 and 5, on the other hand, use $\hat{r}[t] = r[t]$ and so do not require the computation of predicted reinforcement.

The reinforcement-comparison mechanism of Algorithms 10 and 11 is based on the idea that on delayed reinforcement tasks, the prediction of reinforcement at time $t+1$ should depend on all past stimuli rather than on just the stimulus at time t. These algorithms compare primary reinforcement with a trace of the past predicted reinforcement levels:

$$\hat{r}[t+1] = r[t+1] - \bar{p}[t+1]. \tag{18}$$

Algorithm 10 uses $\hat{r}$ as given by (18) both to update the action-association vector $\vec{w}$ by (13) and the reinforcement-association vector $\vec{v}$ by (16).

Experiment 1: The Effect of Delay

In this experiment we returned to nonassociative (or reinforcement-only) tasks to illustrate how severely delayed reinforcement can influence the learning process. Each task is an unbiased continuous-reinforcement task. When the correct action (Action 1) is taken, the expected value of the reinforcement is $+.1$, otherwise it is $-.1$. In either case, the reinforcement is chosen from a normal distribution of standard deviation $\sigma_r = .1$.

We simulated seven tasks that differ only in the extent of the delay between the time of an action and the time of the delivery of the resultant reinforcement. Delays of 0, 5, 10, 15, 20, 25, and 30 time steps were used, where a delay of 0 means that $r[t+1]$ is due to $y[t]$, as in the tasks of preceding sections, and a delay of 5 means that $r[t+1]$ is due to

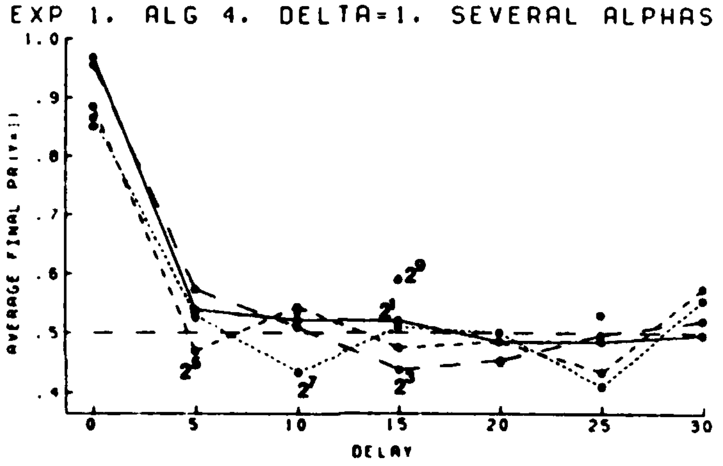
$y[t-5]$, etc.

A major purpose of this experiment was to compare the difficulty of tasks as a function of the length of the delay between action and reinforcement. Therefore, care was taken regarding the number of steps of each run. If each run were allowed to last 15 steps, then the task with a delay of 0 would include 15 reinforcements relevant to the actions taken on the run, whereas the tasks with nonzero delays would include fewer. Tasks with delays of 15 or more would include no reinforcements relevant to the actions taken. To ensure that the same number of relevant reinforcements were delivered on all tasks, each task's runs were extended by the length of the task's delay. The runs of the task of zero delay ran 15 steps, those of the task of delay 5 ran 20 steps, etc. For those runs with a nonzero delay, the reinforcement values for some of the first steps (as many as the length of the delay) were not influenced by any action and were set to zero.

Only Algorithms 4 and 5 were used in Experiment 1. As discussed above, these two algorithms are straightforward extensions of the like-numbered algorithms of the Section 4 to include exponentially decaying eligibility traces. If $\delta = 1$, then Algorithms 4 and 5 of the two sections would be identical. For each task, i.e., for each delay value, and for each of these two algorithms, 200 runs were simulated for each combination of a range of values for the α and δ parameters. The α values used were the powers of two from 2^{-1} to 2^{13}, and the δ values used were the powers of 2 from 2^{0} to 2^{-6}.

The algorithms listed in Table 5 are all written as if they are associative-learning algorithms in that their eligibility factors include x_i, making them dependent on the stimulus presented. For the nonassociative experiments of this section, we assume that the vectors ($\vec{w}$, $\vec{v}$, and $\vec{x}$) are of length 1, and that the single stimulus component x[t] is always 1. Under these assumptions, the associative forms of the learning algorithms reduce to the corresponding nonassociative forms.

At the end of each run the probability of a correct choice on the next step was computed. The average of this probability over the 200 runs, with a given task and algorithm, is the measure of performance plotted in the graphs and discussed below. By looking selectively at different parts of the large array of data generated by this experiment, various issues regarding the effect of delayed reinforcement on learning can be highlighted. The first issue is how well the algorithms without eligibility traces performed on the delayed-reinforcement tasks. Recall that when $\delta = 1$ there are, in essence, no traces, and the algorithms are the same as those of the preceding sections. Figure 23 plots performance versus delay for Algorithms 4 and 5 with $\delta = 1$. In both cases the data are shown for a few representative values for α ($\alpha = 2^1, 2^3, 2^5, 2^7$, and 2^9). The data for Algorithm 4 are in the upper graph and the data for Algorithm 5 are in the lower graph. Note how rapidly performance decreases as the delay increases. As one would expect, for any nonzero delay these traceless algorithms performed only at, or very near, the chance level.



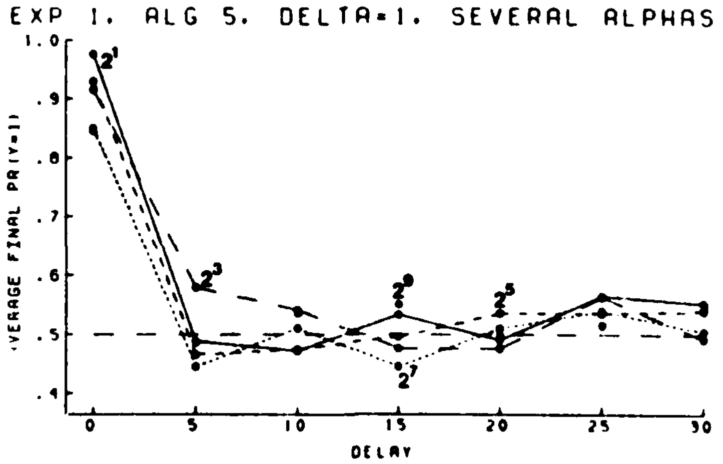


Figure 23

Performances of Algorithms 4 and 5 with no Eligibility Traces on Delayed-Reinforcement Tasks.

With exponentially decaying eligibility traces, the performances of Algorithms 4 and 5 degrade more slowly as the delay length increases. Figure 24 presents the same data as in Figure 23, but from simulation runs with $\delta = 2^{-6}$. Note that even with traces, the performances of these algorithms drop sharply as the delay between action and reinforcement increases. Traces can make learning possible with delayed reinforcement, but they cannot prevent it from becoming much slower. Also note that although traces increase performance on the tasks with delayed reinforcement, they tend to decrease performance on the immediate reinforcement task. This tradeoff is seen most clearly in Figure 25 which shows performance as a function of δ. Each curve shows the performance of the algorithm with $\alpha = 2^3$ for the particular value of δ that is marked near the curve. The use of longer traces (smaller δ's) enabled considerable learning to take place even at long delays. Note that the low-δ versions of each algorithm performed best when reinforcement was delayed, and the high-δ versions performed best when reinforcement was immediate.

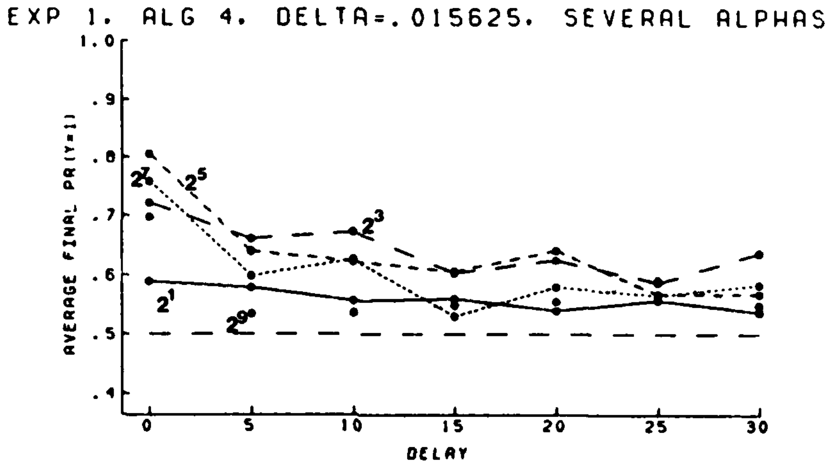
Experiment 2: Sooner Is Better

According to the recency heuristic, the sooner a reinforcing event follows an action the larger its effect in encouraging the action to recur. This heuristic involves a danger of weighting quick reinforcement more heavily than is appropriate. Experiment 2 was designed to demonstrate this danger for the recency heuristic as embodied in eligibility traces.

Experiment 2 involved a single continuous-reinforcement task in which the reinforcement signal is defined by

$$r[t+1] = .1y[t] + .2(1 - y[t-2]),$$

where r is the reinforcement signal, and $y[t] \in \{0,1\}$ is the action selected at time t (defined to be 0 for $t \leq 0$). Both Actions 1 and 0 have positive influences on subsequent reinforcement, but whereas Action 0 has a $+.1$ influence on the immediately following reinforcement, Action 1 has a $+.2$ influence on the reinforcement 2 steps later. If these two influences coincide, then they add, and the resultant reinforcement is $+.3$. If neither



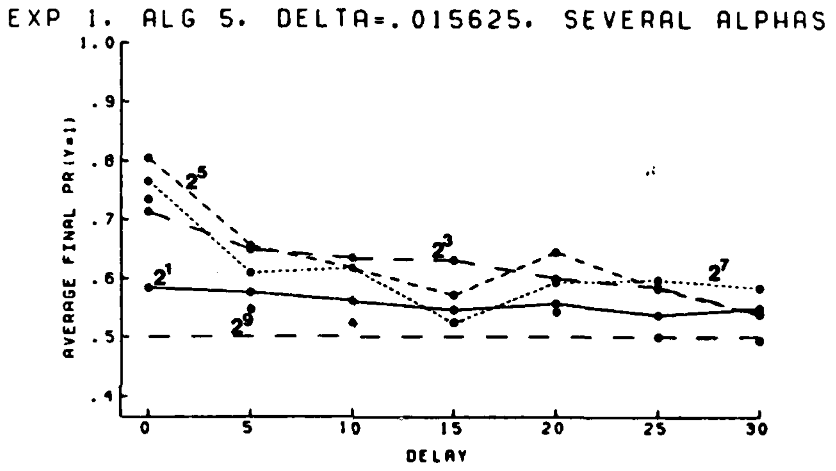


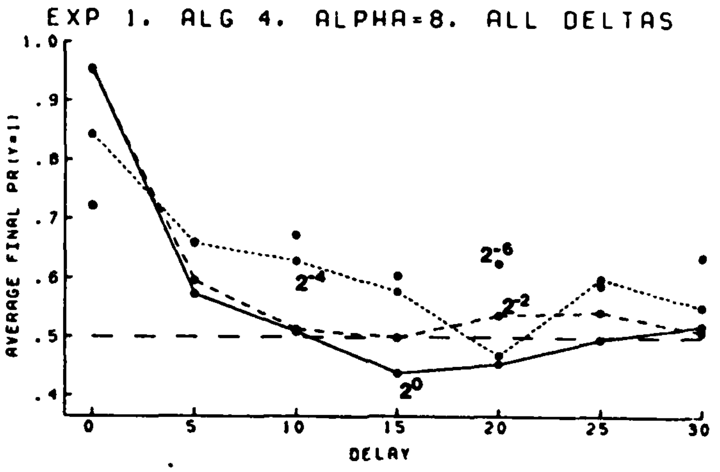
Figure 24

Performances of Algorithms with Exponentially Decaying Eligibility Traces on Delayed-Reinforcement Tasks.

influence applies to a reinforcement value, then it is 0. There is no randomness in the reinforcement signal in this task.

If Action 1 were selected every time step, a reinforcement of +.2 would occur every time step, and if Action 0 were selected every step, a reinforcement of +.1 would occur on every step. Action 1 is clearly the correct action in the long run, but in the initial stages of learning when Actions 1 and 0 are both being tried intermittently, Action 0 has some advantage because the reinforcement it causes is delivered more quickly than that caused by Action 1.

Does this asymmetry regarding the speed with which reinforcement is delivered se-



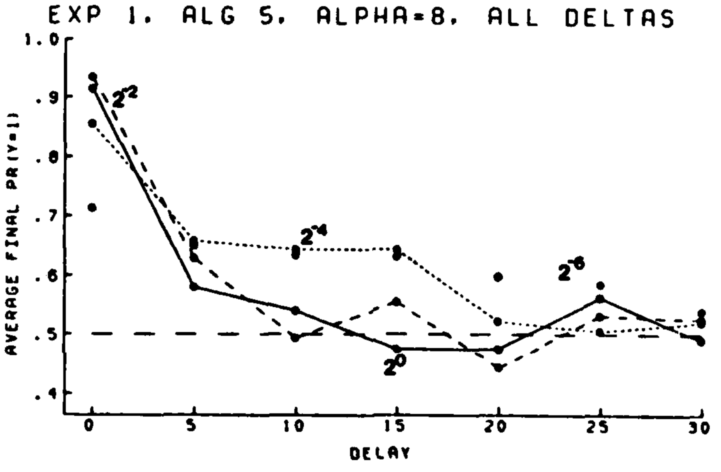


Figure 25

Performances of Algorithms with Fixed Eligibility Decay Rate as Functions of Task Delay.

riously disrupt either the speed of learning or the ability to select the better action for some algorithms? To measure the extent of disruption, performance on this task must be compared with that on a task without the asymmetry. One reasonable such "control task" is one with the same reinforcement levels for the two actions, +.1 and +.2, but with delays both equal to the length of the longer delay in the asymmetrical task (2 time steps). This control task differs from the asymmetrical one only in that one delay has been lengthened from 0 to 2 steps. Since the results of Experiment 1 show that lengthening the delay under these conditions normally makes learning more difficult, if an algorithm performs better on this control task than it does on the asymmetrical task, then it must be due to the removal of a harmful effect of the asymmetry.

For both the control task and the asymmetrical task, simulations were run for all 6 algorithms with each combination of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-5} to 2^{3}, and the δ values used were the powers of two from 2^{0} to 2^{-6}. For each task, algorithm, and parameter setting, 500 runs of exactly 200 steps were simulated. At the end of the 200 steps of each run, the probability $\pi[201]$ of performing the correct action (Action 1) on the next step was recorded. This probability was averaged over the 500 runs to yield a measure of performance on each task for each algorithm at each parameter setting.

Figure 26 shows the highest performance level achieved at any of the values of α tried for each task, algorithm, and δ value. The upper group of plots are all due to performances on the control task, and the lower group of plots are all due to performances on the asymmetrical task. The horizontal dashed line indicates the performance level that would have been attained if actions had been selected totally at random throughout the experiment. This was also the initial performance level.

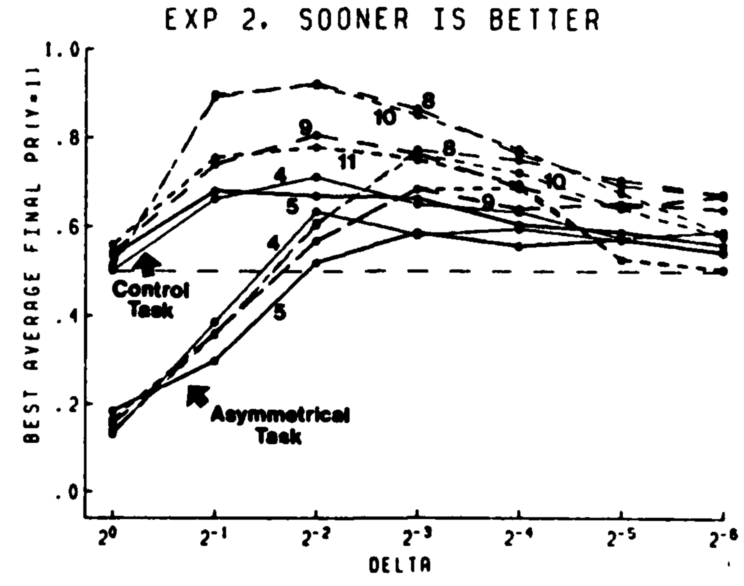


Figure 26

Comparison of Algorithm Performances on Tasks with and without Delay Asymmetries.

All algorithms performed significantly worse on the asymmetrical task than they did on the control task. The best performance of every algorithm on the control task is significantly better than its best performance on the experimental task. In addition, at each of the first four δ values, where each algorithm achieved its best performance, every algorithm performed significantly better on the control task than on the asymmetrical task. This strikingly poor performance of all algorithms on the asymmetrical task compared to their performance on the control task illustrates how debilitating the asymmetry can be.

For $\delta = 1$ and $\delta = \frac{1}{2}$ (the two highest δ values), every algorithm performed significantly worse than the chance level. This indicates that algorithms performed poorly on the asymmetrical task not just because they learned slowly, but because they actually learned to choose the wrong action. In these cases better performance would have been attained with a learning constant $\alpha = 0$, i.e., with no learning at all.

It is possible to mathematically determine the "critical value" for δ, above which the incorrect action is learned, and below which the correct action is learned. As discussed earlier, the eligibility of an action decays as $(1-\delta)^k$, where k is the number of time steps since the action was selected. Since the effect of reinforcement on an action is proportional to both the size of the reinforcement and the eligibility of the action, the effect of a reinforcement of size r delayed by k time steps is $r(1-\delta)^k$. Whether a quick, small reinforcement, or a slow, large reinforcement is the more effective depends on which results in a larger value for $r(1-\delta)^k$. The "critical value" for δ, therefore, is that for which $r(1-\delta)^k$ is equal for the two reinforcements. For the reinforcement sizes and delays in Experiment 2, the "critical value" is that δ for which:

$$.2(1-\delta)^2 = .1(1-\delta)^0.$$

Solving for δ yields $\delta = 1 - \sqrt{1/2} \approx .293$. Since the first two δ values used in Experiment 2 are above this value, and these are the cases in which the incorrect action was learned, this value is consistent with the simulation results.

The primary result of this experiment is that *all* of these algorithms performed poorly in the face of different delay lengths for different actions. This experiment also indicates some significant differences between the performance level of the various algorithms. In particular, the reinforcement-comparison algorithms and the algorithms using eligibility terms involving $y[t] - \frac{1}{2}$ performed better than those without these characteristics. However, we defer discussion of the relative abilities of these algorithms to the next 2 experiments, which make systematic comparisons.

Experiment 3: Comparison of Algorithms on Nonassociative Problems

Experiment 3 was essentially a repetition of the three continuous-reinforcement, nonassociative tasks examined in Section 3, only this time with delayed reinforcement. The intent, as in Section 3, was to compare the performances of the various algorithms.

On all 3 tasks the reinforcement is delayed by 5 time steps for either action. On all tasks Action 1 is the better action. The tasks are called "high," "low," and "middle" tasks, according to their distribution of possible reinforcement values. On the high task, selection of Action 1 results in reinforcement 5 steps later of $+.2$, whereas selection of Action 0 results in reinforcement of $+.1$. The *reinforcement* values for the low task are $-.1$ and $-.2$, and for the middle task, $+.05$ and $-.05$, for Actions 1 and 0 respectively. In all cases the reinforcement depends deterministically on the action chosen.

For all three tasks, simulation runs were performed for all 6 algorithms with each combination of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-3} to 2^{9}, and the δ values used were the powers of 2 from 2^{0} to 2^{-7}. For each task, algorithm, and parameter setting, 200 runs of exactly 200 steps each were simulated. At the end of the 200 steps, the probability $\pi[201]$ of performing the correct action (Action 1) on the next step was computed and recorded. This probability, averaged over the 200 runs, is the performance measure for each combination of algorithm and parameter setting on each task.

Figures 27, 28, and 29 summarize the data from a single task of Experiment 3. Each point in these graphs represents the best performance level of a particular algorithm at a particular δ value on a particular task. These performance levels are those with the α

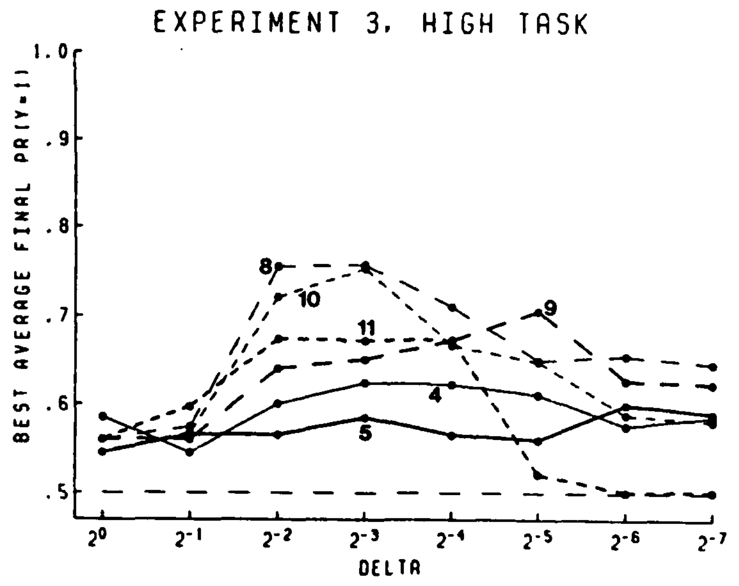


Figure 27

Performances of Algorithms on the *High* Continuous-Reinforcement Nonassociative Task.

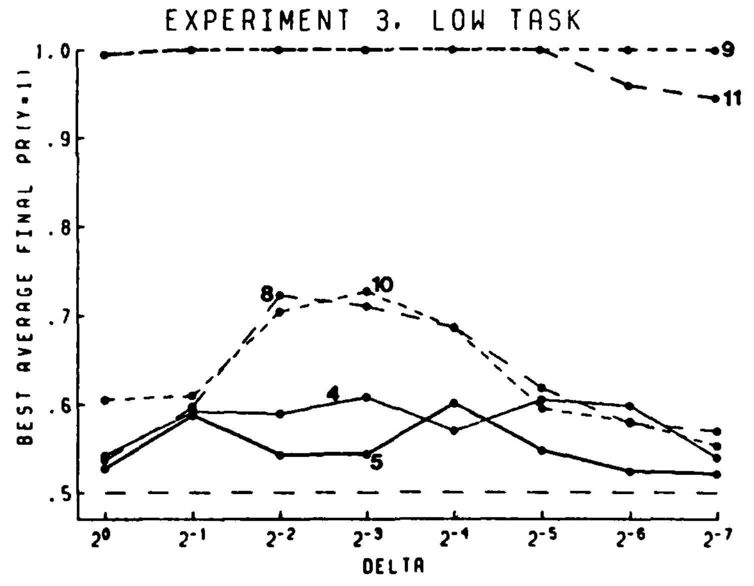


Figure 28

Performances of Algorithms on the *Low* Continuous-Reinforcement Nonassociative Task.

value for each task, algorithm, and δ value that resulted in the best performance.

On the high and low tasks all the reinforcement-comparison algorithms performed better than all the algorithms that do not employ reinforcement comparison. On the

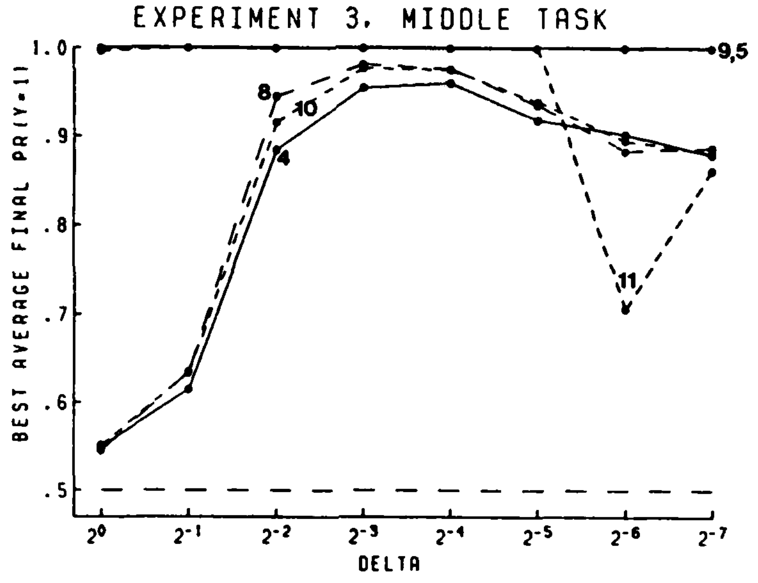


Figure 29

Performances of Algorithms on the *Middle* Continuous-Reinforcement Nonassociative Task.

middle task, Algorithm 5, a non-reinforcement-comparison algorithm, performed better than reinforcement-comparison Algorithms 8 and 10. On this task, Algorithm 5 and the other two reinforcement-comparison algorithms, Algorithms 9 and 11, all performed at the optimal level, so that any performance difference between them was lost in a ceiling effect.

Recall that with $\delta = 1$, all algorithms become identical to those developed for immediate reinforcement and discussed in preceding sections. The good performance of the reinforcement-comparison algorithms, together with the fact that in all cases their best performance was attained at a δ value less than 1, provide evidence that the new reinforcement-comparison algorithms have been generalized properly for application to tasks with delayed reinforcement.

The performances of the algorithms using $y[t] - \pi[t]$ in their eligibility factors (Algorithms 5, 9, and 11), as contrasted to those using $y[t] - \frac{1}{2}$ in their eligibility factors (Algorithms 4, 8, and 10), are generally similar to those described in Section 3. On the low and middle tasks, those algorithms using $y[t] - \pi[t]$ were clearly superior in most cases. On high task, the $y[t] - \frac{1}{2}$ algorithms were sometimes clearly best. This pattern is similar to that found in Section 3 with immediate-reinforcement tasks. In both cases the combination of reinforcement comparison and $y[t] - \pi[t]$ eligibility tended to improve performance in most cases.

Experiment 4: Reinforcement Delay Asymmetry in Associative Learning

The experiments Section 4 measured the influence on performance of four different kinds of asymmetry in the treatment of stimuli on associative-learning tasks and attempted to relate performance differences among algorithms to features of those algorithms. Delayed reinforcement creates the possibility for another type of asymmetry between stimuli: Reinforcement for an action chosen in response to one stimulus may be delayed more than reinforcement for an action chosen in response to a second stimulus. Experiment 4 adds this *reinforcement-delay asymmetry* to those of the preceding section.

Reinforcement-delay asymmetry is not to be confused with the asymmetry studied in Experiment 2 of this section. The asymmetry of Experiment 2 is an *action* asymmetry rather than a *stimulus* asymmetry. In Experiment 2's task, the actions are treated differently, whereas in Experiment 4's task, as in the tasks of Section 4, the stimuli are treated asymmetrically.

As in most of the tasks of the preceding section, Experiment 4's task involves two stimuli presented at random with equal frequency. The two stimuli are similar and yet correspond to different optimal actions. The two stimuli are

$$\vec{x}^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{x}^2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

Actions chosen in response to $\vec{x}^2$ influence reinforcement on the next step (i.e., immediately, no delay), and actions chosen in response to $\vec{x}^1$ influence reinforcement after a delay of 4 steps. In either case, actions have either a $+.1$ influence on subsequent reinforcement or a $-.1$ influence. In response to $\vec{x}^2$, Action 0 is the action with the positive influence, whereas in response to $\vec{x}^1$, Action 1 is the action with the positive influence. If influences from two actions in response to two different stimuli both influence the same reinforcement value, their influences summate, e.g., if $y[t] = 0$ is selected in response to $\vec{x}[t] = \vec{x}^2$, and $y[t-4] = 1$ is selected in response to $\vec{x}[t-4] = \vec{x}^1$, then $r[t+1]$ is $+.2$. If a reinforcement is not influenced by previous action, it remains zero. In the tasks of this experiment, reinforcement is computed deterministically from the actions selected.

Simulations were performed for all 6 algorithms with all combinations of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-5} to 2^6, and the δ values used were the powers of 2 from 2^0 to 2^{-7}. For each algorithm and parameter setting, 200 runs of exactly 150 steps were simulated. At the end of each run, the final weight vector $\vec{w}[151]$ was recorded. From this, the probability of selecting each action in response to each stimulus, and the expected influence on reinforcement of the next action, was computed as described in Section 4. The expected influence of the next action on subsequent reinforcement is a good measure of performance on a particular run. This measure was averaged over the 200 runs to produce a performance measure for each algorithm with each parameter setting.

Figure 30 summarizes the data from Experiment 4, showing only the performance levels for the α value associated with maximum performance for each algorithm and δ value. Algorithms 9 and 11 attained the highest performance levels, and were especially superior at low δ values. This result suggests that reinforcement-comparison and the use of an eligibility term including $y[t] - \pi[t]$, as opposed to $y[t] - \frac{1}{2}$, produce better performance on this task than the alternative mechanisms.

Conclusion

Reinforcement learning tasks with delayed reinforcement are much more difficult than corresponding tasks with immediate reinforcement because of the uncertainty introduced as to which actions cause which reinforcing events. The algorithms studied in previous sections were designed for immediate reinforcement tasks and are not effective on delayed reinforcement tasks. However, by adding exponentially-decaying eligibility traces, these algorithms can be modified so as to greatly improve their performance on delayed-reinforcement tasks. In selecting the durations of these traces one must accept a tradeoff between performance on short-delay tasks and performance on long-delay tasks. Trace

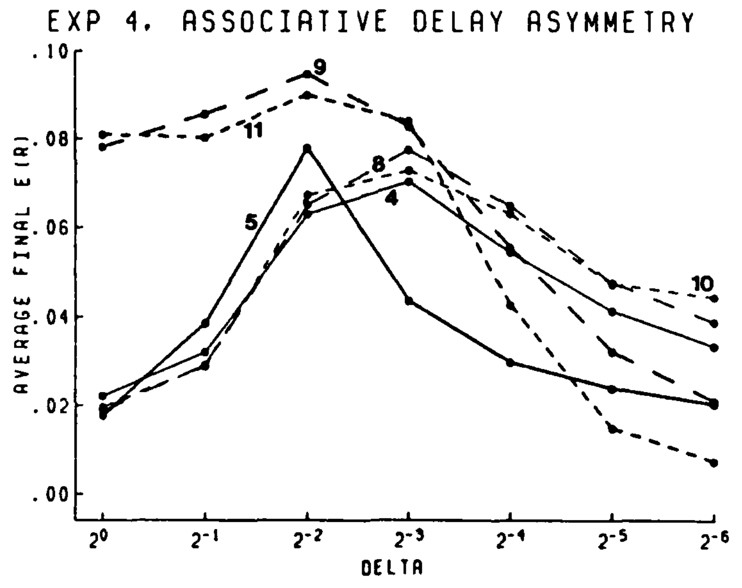


Figure 30

Performances of Algorithms on a Task with Associative-Delay Asymmetry.

durations optimal for long delays result in suboptimal learning rates on tasks with short delays. Whatever trace durations are used, learning is always slower with longer delays than with shorter delays. Again, we are assuming that no *a priori* knowledge of delay length can be used in the algorithm's design.

For algorithms with eligibility traces based on a recency heuristic, delayed reinforcement is less effective than immediate or less-delayed reinforcement. With such algorithms, a small immediate reinforcement may be more effective in reinforcing behavior than a larger delayed reinforcement. Experiment 2 showed that all algorithms considered here, even those using reinforcement-comparison techniques and $y[t] - \pi[t]$, are susceptible to this problem. On nonassociative tasks with equal delays for both actions, the algorithms introduced in this section perform similarly to the algorithms on the corresponding immediate-reinforcement tasks of Section 3. Overall, there are advantages to both reinforcement-comparison techniques and the use of a $y[t] - \pi[t]$ eligibility mechanism.

Finally, delayed reinforcement opens the possibility of a new type of asymmetry between the treatment of stimuli in associative learning — the delay between action and reinforcement may be different for the same actions taken in response to different stimuli.

Experiment 4 contrasted the various algorithms in their ability to deal with this asymmetry on a test analogous to those described in Section 4. As in Section 4, reinforcement-comparison and $y[t] - \pi[t]$ algorithms performed best.

The recency and frequency heuristics studied in this section are conventionally thought of as "general but weak" methods, as opposed to the knowledge-rich "specific but powerful" methods that are central to much of modern AI. Our intention in studying these heuristics has not been to dispel this conventional assessment, but to establish and detail it experimentally. The results reported in this section are not surprising ones; they are roughly what one might expect. Their importance is primarily in confirming and demonstrating what would otherwise be merely hypothesis and presumption, and in detailing the magnitude, extent, and generality of the various phenomena.

Although delayed reinforcement is a necessary property of any task with opportunity for secondary reinforcement, tasks of this section were carefully designed to offer no such opportunities. All tasks of this section either had no neutral stimuli or else neutral stimuli that were presented at random. Opportunities for secondary reinforcement arise only when the actions of the learning system affect subsequent neutral stimuli (which can be made non-neutral by the secondary-reinforement mechanism). These studies of delayed reinforcement without secondary reinforcement provided a point of departure for the study of secondary reinforcement reported in the next section.

Section 6
SECONDARY REINFORCEMENT

Introduction

In this section we provide a heuristic justification for an algorithm for reducing the severity of temporal credit assignment. This algorithm constructs a reinforcement signal that is of "higher quality" than the primary reinforcement signals available from the learning system's environment. We call this algorithm the *adaptive heuristic critic* (AHC) algorithm. It can be regarded as a mechanism for generating secondary reinforcement and is an extension of the model of classical conditioning presented by Sutton and Barto (1981) and Barto and Sutton (1982). This model and the AHC algorithm were influenced by Klopf's (1972, 1982) concept of "generalized reinforcement." According to this idea, all (or at least many) inputs to a neuronlike element can transmit reinforcing signals. Among other things, this implies that as an element adjusts its weights, it not only alters its input/output mapping but also redefines its evaluation signal. Properly implemented, such a mechanism can construct a reinforcement signal that is more useful than the initial, or primary, reinforcement signal. The AHC algorithm is the result of our effort to study this type of process. The AHC algorithm also turns out to be closely related to the algorithm used by Samuel (1959) in his checkers playing program. We begin the justification of this algorithm by proposing that an ideal reinforcement signal should have certain properties.

The *ideal reinforcement signal* should be positive in value whenever the immediately preceding action was better than average for the situation in which it was taken, and negative whenever that action was worse than average. Its magnitude should indicate how much better or worse than average the immediately preceding action was. By the *value* of an action, we mean a measure of its expected effect on future values of the primary reinforcement signal. The ideal reinforcement signal would indicate to the learning system the value of the given action relative to the sum of the values of all possible actions, each weighted according to the given action's probability of being chosen in the given situation. Since these probabilities depend on the current state of the learning system, so does this average, and consequently the ideal reinforcement signal would also depend on the state of the learning system. In fact, since the ultimate effect of an action on primary reinforcement will usually be influenced by subsequent actions, the value of an action will also in general depend on the current state of the learning system. Since the learning system changes as it accumulates experience, the ideal reinforcement signal also changes.

For example, suppose there is a problem with 3 actions, with the values 0, .8, and .9 when taken in a particular situation by a learning system in a particular state. In the early stages of learning, when the 3 actions are each selected equally often, the ideal reinforcement signal will be positive for the latter two actions, and negative for the first. As learning progresses, and the probability of selecting the first actions falls to zero, the ideal reinforcement signal will change so as to be negative for the second action, remaining positive only for the third action. One could argue that in order to satisfy one sense of the word "ideal," the ideal reinforcement signal should be a constant function of the actions and should always be positive only for the best of the actions, in this case only for the third action. Such a reinforcement signal might cause perfect performance to be attained more quickly, but only at the cost of sacrificing interim performance. Our intent is to define the ideal reinforcement signal so that it stimulates the maximization of cumulative performance.

To further clarify what we mean, consider an example using the game of chess. Suppose that from a particular position the learning system can force checkmate by playing first move y^1 and then move y^2, but that if it plays y^1 and then misses the winning move y^2, it will lose the game. Suppose further that with the learning system's current knowledge base, the latter is exactly what would happen if it chooses y^1. Suppose finally that if the learning system chooses some move other than y^1, then it has a better than even chance of winning the game. In one sense, y^1 is a good move in this position, since it is the first move in a sequence of moves that guarantees a win. On the other hand, *with the current knowledge base,* y^1 is not a good move because it results in the immediate loss of a game that might otherwise be won. Our intent is to define the value of a particular move in the latter, local sense that would label y^1 a poor move. We wish to consider each move in isolation from the others and ask "Other things left as they are, does selecting this move make the prospects better or worse?"

The ideal reinforcement signal would be an improvement over the primary reinforcement signal in three ways. These are, in decreasing order of importance to the work presented here:

Immediacy -- Some effects of the choice of an action on subsequent primary reinforcement may be delayed. In the ideal reinforcement signal all delayed effects are "brought closer to the present" so that they are effective immediately after the action is taken. Immediate reinforcement is an improvement over delayed reinforcement because there is no uncertainty as to which action caused it.

Comparison with a reinforcement standard — A particular reinforcement level is not good or bad in itself, but only in comparison with other reinforcement levels that might have been received had the learning system or the environment behaved differently. Since the ideal reinforcement signal would rate actions that are better or worse than average as being positive and negative respectively, it would provide direct information as to whether the selected action is a good or a bad selection. The ideal reinforcement signal would remove the burden from the learning system of comparing reinforcements with a *reinforcement standard* and for determining what that standard should be.

Increased reliability — The ideal reinforcement signal would be a relative measure of how good or how bad the selected action is *on the average.* Whereas the primary reinforcement signal can be different each time a particular action is taken in a particular situation, due either to random aspects of the environment, or to variations in subsequent action selections, the ideal reinforcement signal would provide a reliable measure that takes into account all possible variations and their likelihoods.

The concept of the ideal reinforcement signal is similar to that of the ideal evaluation function for guiding a state-space search or the search through a game tree. In either case this ideal is rarely obtainable but can only be approximated. In either case one looks to heuristics, either learned or provided *a priori*, to build the approximation. A *heuristic reinforcement signal* is a reinforcement signal generated internally by a reinforcement-learning system, which it uses instead of the primary reinforcement signal. The idea is that the heuristic reinforcement signal is more similar to the ideal reinforcement signal than is the primary reinforcement signal. To improve on the primary reinforcement signal, a learning system may use either *a priori* knowledge or knowledge gained from its past experience. In the research described here, we considered only the latter case, and assumed that all available knowledge had already been built directly into the primary reinforcement signal.

Given the limited information a learning system receives about the state of its environment, how good a heuristic reinforcement signal is actually possible? The *ideal realizable reinforcement signal* is positive at the first indications from the environment that higher than average primary reinforcement is forthcoming, and negative at the first indications that lower than average primary reinforcement is forthcoming. To the extent that the environment provides such information, the size of the signal indicates the amount by which the forthcoming reinforcement is higher or lower than average.

In some cases the first indication of forthcoming primary reinforcement is the primary reinforcement itself. In this case the ideal realizable reinforcement signal is an improvement over the primary signal only by virtue of providing a performance standard, and not by improving immediacy or reliability. In other cases, neutral stimuli provide the first indications of unusually high or low forthcoming reinforcement, and the latter two improvements are realizable as well. It is the ideal realizable reinforcement signal that one can most fruitfully attempt to emulate by a heuristic reinforcement signal.

The Adaptive Heuristic Critic Algorithm

In the adaptive heuristic critic (AHC) algorithm, a linear-mapping approach is used to associate predictions of forthcoming reinforcement with stimuli. Stimuli are represented as vectors of n components. Associated with each stimulus component x_i, $1 \leq i \leq n$, is a memory variable v_i indicating the extent to which the presence of the stimulus component indicates that unusually high or low reinforcement is forthcoming. As in Sections 4 and 5, we call the vector $\vec{v}[t] = (v_1[t], v_2[t], \ldots, v_n[t])$ the *reinforcement-association vector*. The net prediction of forthcoming reinforcement associated with a stimulus vector is the sum of the stimulus components weighted by the components of the reinforcement-association vector. We define the prediction of reinforcement forthcoming after time t, made by the learning system at time s (i.e., using $\vec{v}[s]$) as:

$$p^s[t] = \sum_{i=1}^{n} v_i[s]x_i[t]. \tag{19}$$

The reinforcement-association vector should be formed so that $p^s[t]$ is a good estimate of forthcoming reinforcement. It follows, therefore, that a signal that is a good heuristic reinforcement signal would provide an excellent basis for correcting estimates of $p^s[t]$ by changing $\vec{v}$. The following update rule accomplishes this:

$$v_i[t+1] = v_i[t] + \beta\, \hat{r}[t+1]\bar{x}_i[t+1], \tag{20}$$

for $1 \leq i \leq n$ and $t = 0, 1, \ldots,$ where $\hat{r}[t+1]$ is the heuristic reinforcement signal's value at time $t+1$, β is a positive constant, and $\bar{x}_i[t+1]$ denotes the value at $t+1$ of a trace of x_i, i.e., a weighted average of the past values of x_i with the more recent values weighted more heavily (see Section 5). If $\hat{r}[t+1]$ is positive (negative) this equation increases (decreases) the components of $\vec{v}$ that contributed to past predictions $p^s[t]$, as indicated by their $\vec{x}$ components having been large in the recent past. If the same stimulus sequence were presented again, this process would result in those predictions being higher (lower) earlier in the sequence. The overall result is that the positive (negative) $\hat{r}$ event is shifted earlier in time, and the heuristic reinforcement signal does a better job of giving the earliest possible indication of changes in forthcoming reinforcement.

The above argument relies on the heuristic reinforcement signal working properly. The following definition of the heuristic reinforcement signal is used in the AHC algorithm:

$$\hat{r}[t+1] = r[t+1] + \gamma\, p^t[t+1] - p^t[t], \tag{21}$$

where γ, $0 \leq \gamma \leq 1$, is a scalar parameter. Equations 19, 20, and 21 constitute the AHC algorithm. We now turn to the justification of (21).

Let r^* denote a hypothetical ideal reinforcement signal. Its value at time $t+1$ indicates how much better or worse off the learning system is because of the particular action selected at time t. Better or worse is defined in terms of the effect of the action choice on subsequent primary reinforcement signal values $r[t+k]$, $k \geq 1$. The action $y[t]$ at time t may affect primary reinforcement at any combination of later times. These separate effects must somehow be combined in r^* to give a measure of the action's effect. Perhaps the simplest approach would be to sum these effects if they could be determined. This leads one to attempt to define the ideal reinforcement signal as

$$r^*[t+1] = \sum_{k=1}^{\infty} E\{r[t+k] \mid y[t]\} - E\{r[t+k]\}, \tag{22}$$

where the expectations are conditional on the structure and state of the environment and of the learning system. One problem with (22) is that the infinite sum may not be defined. Another is that although this definition seems a natural one, for certain special classes of tasks it is not. In the following section we consider the class of ***time-blind tasks***, for which a different definition of r^* is appropriate.

The sum in (22) may not converge either because it is unbounded or because the sequence of partial sums includes infinite subsequences not all of which converge to the same limit. Figure 31a shows the state-transition structure of an environment in which the former occurs, and Figure 31b shows an environment in which the latter occurs, for the action selected in leaving the state labeled A. These cases are problematic for the definition in (22) but seem not to be problematic on an intuitive basis. The reinforcement given to the action leaving State A is unimportant because the state will never be reentered. In many cases these definitional problems can be eliminated by weakening the requirement of convergence of (22) to that of summability, e.g. Césaro summability:

$$r^*[t+1] \equiv \lim_{N\to\infty} \sum_{M=1}^{N} \frac{1}{N} \sum_{k=1}^{M} \Big[E\{r[t+k] \mid y[t]\} - E\{r[t+k]\}\Big]. \tag{23}$$

Since we have placed no restrictions on the tasks or the learning systems, it is possible that this limit is also undefined. In general it may be necessary to make a slightly different definition of r^* depending on the class of tasks or learning systems being considered. For the purposes of the rest of this subsection, we assume that the limit in (23) exists.

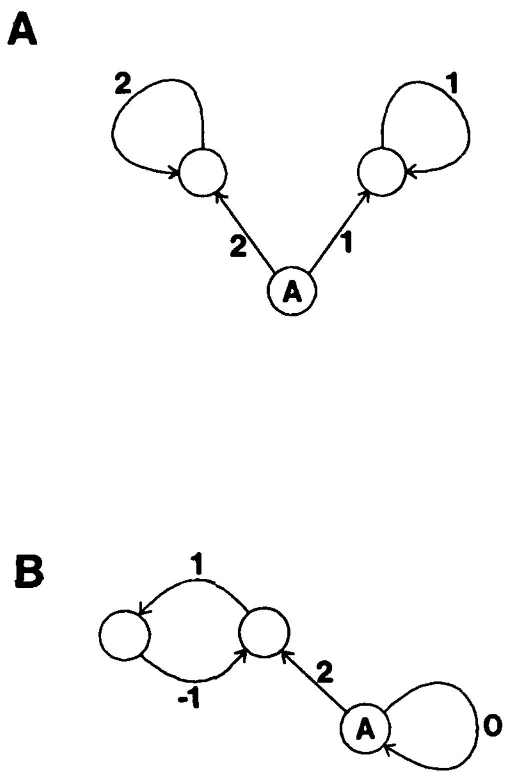


Figure 31

Two Examples of Environments that are Problematic for Definitions of the Ideal Reinforcement Signal.

Let us call the sequence formed by $E\{r[t+k] \mid y[t]\} - E\{r[t+k]\}$ for successive values of k the ***difference sequence***. Each element of this sequence is the difference between the expected value of reinforcement at some later time with and without the selection of $y[t]$. In a typical case we might expect $y[t]$ to influence reinforcement and the state of the environment for a while, but that for large values of k the expected values of reinforcement in the two terms are equal, and the corresponding elements of the difference sequence are zero. In such a case the sum of all elements of the sequence would be finite, and its value would be r^*.

One possible approach to approximating r^* would be to maintain a memory variable whose value would be the current estimate of the sum of the difference sequence. A major drawback to this approach is that these differences are never directly observable. Each is the difference between two expected values of $r[\tau]$, both for some time τ, but conditional on different circumstances. The actual values of r provide an estimate of the expected value for the circumstances that actually occur, but the other circumstances remain hypothetical. A second drawback to this approach is that an estimate of the sum of a difference sequence would be needed for every combination of action and environmental state in which the action was taken, which adds up to a great many memory variables to update and store.

Another approach, which is used in the AHC algorithm, is to estimate the sum of the difference sequence by constructing separate estimates of the sum of the $E\{r[t+k] \mid y[t]\}$ terms and the sum of the $E\{r[t+k]\}$ terms, and then to subtract the two estimates. The logic of this approach can be illustrated by rewriting (23) as a difference of two sums:

$$r^* = \lim_{N\to\infty} \sum_{M=1}^{N} \frac{1}{N}\Big[\sum_{k=1}^{M} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{M} E\{r[t+k]\}\Big]. \tag{24}$$

One might hope that the following is an equivalent expression:

$$r^* = \lim_{N\to\infty} \sum_{M=1}^{N} \frac{1}{N} \sum_{k=1}^{M} E\{r[t+k] \mid y[t]\} - \lim_{N\to\infty} \sum_{M=1}^{N} \frac{1}{N} \sum_{k=1}^{M} E\{r[t+k]\}. \tag{25}$$

In this approach, one would attempt to estimate both terms of (25) from observed values of r, and then subtract the two estimates to estimate r^*. Unfortunately, (24) and (25) are not equivalent. Although (24) is well-defined, equivalent to (23), and, by assumption, finite, the two major terms of (25) will typically both be infinite. Making finite estimates of these two terms and then subtracting them will yield a meaningless result.

The AHC algorithm's solution to this problem relies on the fact that the elements of the difference sequence tend to be nonzero primarily only if they appear early in the sequence, or, stated in different terms, that for large k the expected value of $r[t+k]$ tends to depend very little on $y[t]$. If we assume that

$$E\{r[t+k] \mid y[t]\} = E\{r[t+k]\},$$

for all k greater than M, then we could rewrite (23) as

$$r^*[t+1] = \sum_{k=1}^{M} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{M} E\{r[t+k]\}. \tag{26}$$

As long as most of the nonzero terms of the sequence occur early, i.e., in the first M terms, this will be a good approximation to the full sum. The advantage of (26) is that both of its sums are finite and can be estimated easily.

The AHC algorithm actually works a bit differently, but the idea is the same. Rather than having an abrupt cutoff at M time steps, succesive elements of the difference sequence are weighted in an exponentially decreasing manner. This avoids problems with "horizon effects," and allows arbitrarily late elements of the difference sequence to contribute to the estimate of r^*, albeit possibly greatly discounted. The AHC algorithm is based on estimating

$$r^*[t+1] \approx \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k]\}, \tag{27}$$

where $0 \leq \gamma < 1$. These sums converge so long as $E\{r[t+k]\}$ is bounded.

γ is called the ***discount rate***. The use of a discount rate effectively assigns greater value to earlier primary reinforcement than later primary reinforcement. By adjusting γ, one controls the extent to which the learning system is concerned with long-term versus short-term goals. It is common in studying Markovian decision problems to introduce such a discount rate by taking the maximization of (27) directly as the goal of learning rather than as an approximation (Derman, 1970; Mine and Osaki, 1970).

The expected values we have discussed thus far are all implicitly conditional on the structure and state of the particular environment and learning system. In the following it is necessary to forego this notational convenience in the case of the states. Denoting the state of the environment at time t as $q[t]$ and the complete state of the learning system at time t as $w[t]$,* we rewrite (27) as:

$$r^*[t+1] \approx \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t], q[t], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}. \qquad (28)$$

At time $t+1$, $r[t+1]$ will be available for use as an estimate of $E\{r[t+1] \mid y[t], q[t], w[t]\}$. Using this approximation, and taking this term outside the first sum in (28) yields

$$r^*[t+1] \approx r[t+1] + \sum_{k=2}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t], q[t], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}. \quad (29)$$

Since $r[t+k]$, for $k \geq 2$, does not depend on $y[t]$ directly, but only through $y[t]$'s effect on $q[t+1]$, the first expected value in the above expression can be rewritten to be conditional only on $q[t+1]$:

$$r^*[t+1] \approx r[t+1] + \sum_{k=2}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t+1], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\},$$

or, making the change of variable $k \rightarrow k+1$ in the first sum:

* Note that in this context $w[t]$ denotes the *complete* state of the learning system. In the case of the AHC algorithm, this includes both the action-association vector $\vec{w}$ and the reinforcement-association vector $\vec{v}$.

$$r^*[t+1] \approx r[t+1] + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+1+k] \mid q[t+1], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}.$$

As mentioned above, in the approach taken by the AHC algorithm, the learning system attempts to directly construct estimates of the two expected values in these equations. The algorithm can only use information actually available to the learning system. The expected values in the above expression are conditional on the state of the environment. Direct information about the state of the environment is not available to the learning system, but stimuli are available that give clues as to the state of the environment. The best a realizable heuristic reinforcement signal can do is approximate expected values on the basis of this stimulus information. Denoting by $x[t]$ the stimulus received at time t, which provides information about the state of the environment $q[t]$ at time t, the best a realizable heuristic reinforcement signal can do is approximate the following quantity:

$$r^*[t+1] \approx r[t+1] + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+1+k] \mid x[t+1], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid x[t], w[t]\}. \tag{30}$$

In order to do this, the learning system maintains an estimate of the expected value of forthcoming discounted primary reinforcement. We will call this the *prediction of forthcoming reinforcement*:

$$p^*[t] \approx \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid x[t], w[t]\}. \tag{31}$$

This is an estimate of reinforcement forthcoming atfter time t made by the learning system at time s. The expected value in (31) depends on the actions the learning system is likely to make on subsequent steps. This is why $w[t]$, the state of the learning system, conditions the expected value. Since the state of the learning system may change at each time step, the expected value above may change at each time step. Because of this, any statistical regularities that the learning system manages to pick up will necessarily be slightly out-of-date as soon as they are determined. For a statistical approach to work, the effect of changes in w on the expected values above must be small, at least over short time periods. If we make this assumption, then we can ignore small changes in w, such as those between $w[t]$ and $w[t+1]$, and approximate (30) by the following heuristic reinforcement signal:

$$\hat{r}[t+1] = r[t+1] + \gamma p^t[t+1] - p^t[t],$$

which is the expression used in the AHC algorithm (c.f. (21)).

Estimates $p^s[t]$ can be constructed in any number of ways. Since these estimates must be associated with stimuli, the remarks of Section 4 regarding various association schemes apply. The AHC algorithm uses the linear-mapping approach given by (19), and uses $\hat{r}$ as an error term to update its estimates by (20). Witten (1977) used the same $\hat{r}$ as the AHC algorithm to update estimates associated with states of the environment. However, he used the independent associations approach and used a quantity different from this $\hat{r}$ as heuristic reinforcement to update action associations.

Special Cases

Analyses similar to that presented above can be performed for specialized classes of tasks. Here we omit these analyses but present the resulting specialized versions of the AHC algorithm for each case. Details can be found in Sutton (forthcoming). We consider three types of tasks: ***time-blind tasks***, ***time-until-failure tasks***, and ***time-until-success tasks***. These three types of tasks are examples of ***episodic tasks***. In these tasks, the environmental interaction can be naturally divided into episodes, where the performance during each episode is dependent only on the behavior during the episode, and where the boundaries between episodes are clearly demarcated. In chess or backgammon, for example, the episodes are games.

Whether or not a task is episodic can be very important for purposes of credit assignment. At the end of each episode the learning system is guaranteed that there will be no further delayed effects of any of the actions taken during the episode. Normally this is not possible; the learning system can not "close the books" on the credit to be assigned to any of its past behavior, for there may always be some further consequences. It is the possibility of such arbitrarily delayed effects that complicates the definition and approximation of the ideal reinforcement signal presented in the preceding subsection.

Time-blind tasks are a particular kind of episodic task in which the goal of learning is defined in terms of performance per episode rather than in terms of performance per time step. In a time-blind task, the goal of learning is "blind" to the duration of the episodes. Most games are time-blind tasks; in chess or checkers, for example, there is, at least in theory, no concern about the length of each game, but only about its outcome.

In most time-blind tasks each possible outcome of an episode can be assigned a definite value, e.g., $+1$, -1, and 0, for win, loss, and draw of a chess game. We generalize this slightly by allowing reinforcement to be delivered throughout the episode. In this case the goal of learning is to maximize the expected value of the sum of the reinforcement received during the episode:

$$E\left\{\sum_{\tau=1}^{m} r[\tau]\right\},$$

where $r[\tau]$ denotes the reinforcement received on the τth time step of the episode, and m denotes the length in time steps of the episode. This expectation is implicitly conditional on the task, the learning system, and the state of the learning system. Tasks in which the outcome of the episode is known only at its end are easily handled within this framework by having the reinforcement for all transitions except the final one be zero. One can argue that under these conditions, the appropriate heuristic reinforcement signal is

$$\hat{r}[t+1] = r[t+1] + p^t[t+1] - p^t[t],$$

which is the same as (21) with $\gamma = 1$.

By a time-until-failure task we mean a task in which the goal of learning is to maximize the expected duration of the episodes. In this case, it can be argued that the heuristic reinforcement signal is:

$$\hat{r}[t+1] = 1 + p^t[t+1] - p^t[t].$$

A time-until-success task is a task in which the goal is to complete each episode in the minimum amount of time. An example is the task of completing a maze. For these tasks, it can be shown that the appropriate heuristic reinforcement signal is

$$\hat{r}[t+1] = -1 + p^t[t+1] - p^t[t].$$

We refer the reader to Sutton (forthcoming) for detailed justification of these variants of the AHC algorithm and a discussion of their relationship to the algorithms of Samuel (1959) and Witten (1977).

Illustration

Figure 32 shows the behavior of the AHC algorithm when it is presented with an input sequence consisting of a temporal series of clearly distiguishable stimuli followed by a brief period of primary reinforcement. Time trajectories of the input variables are shown twice in the upper part of the figure, and synchronous plots of $\hat{r}$ and $p^t[t+1]$ on successive trials from Trial 1 to Trial 10 appear below. This experiment used parameter values of $\beta = .6$, $\gamma = 1$, and $\tau = 5$. The simulation had not quite reached its asymptotic state at the end of the 10 trials shown here. Eventually, $\hat{r}$ became positive only at the onset of the earliest stimulus component, and $p^t[t+1]$ remained constant over the time interval during which stimuli were presented.

Notice how the times of positive $\hat{r}$ gradually moved earlier in the input sequence. First, the latest stimulus components became associated with primary reinforcement, then they acted as the basis for the association of earlier stimulus components. The behavior of the prediction $p^t[t+1]$ followed a similar pattern. Prediction of reinforcement was first made just before the arrival of reinforcement, and then it moved back as far as was possible with

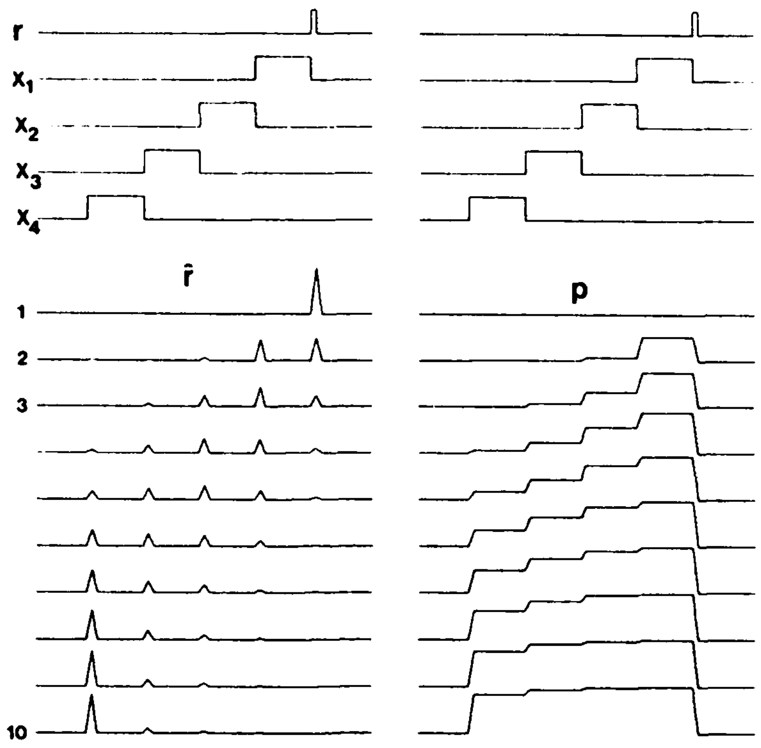


Figure 32

Behavior of the AHC Algorithm for a Temporal Series Stimuli Followed by a Brief Period of Primary Reinforcement.

the available stimulus information. $\hat{r}$ became almost equal to a discrete-time derivitive of p.* With the exception of the last one, the size of each positive blip of $\hat{r}$ very closely matches the size of the corresponding increment in p. The last blip at the time of the primary reinforcement event, reflects a combination of a positive influence due to primary reinforcement, shown alone in the Trial-0 plot, and a negative influence in later trials due to the decrement in p that occurred at this time. In the last trials, these two influences exactly cancelled out.

This simulation experiment corresponds to a classical, or Pavlovian, conditioning experiment with a serial-compound conditioned stimulus (see Kehoe, 1982). Many other

* It would have been exactly this derivitive except for the small changes in the reinforcement-association vector $\vec{v}$ from time step to time step.

simulation experiments with the isolated AHC are described in Sutton (forthcoming). Next we describe the use of the AHC algorithm in a reinforcement-learning task.

Pole Balancing

As a vehicle for illustrating the capabilities of a reinforcement-learning system coupled with the AHC algorithm, we applied a learning system to a control problem. This task is to learn to balance a pole that is hinged to a moveable cart by applying forces to the cart's base. It is assumed that the equations of motion of the cart-pole system are not known and that the only feedback evaluating performance is a failure signal that occurs when the pole falls past a certain angle from the vertical or the cart reaches the end of a track. In formulating this task, we followed the work of Michie and Chambers (1968a, b) and implemented their learning system, called BOXES, in order to serve as a basis of comparison for the performance of our learning system. This illustration is discussed in detail in Barto, Sutton, and Anderson (1983).

Although the system to be controlled is a realistic physical system (even though we simulated it by computer), we were not interested in pole-balancing, but rather in the type of problem our formulation (following that of Michie and Chambers) represents. The sparsity of evaluative feedback creates a genuinely difficult temporal credit-assignment problem. Since the failure signal occurs only after a long sequence of individual control decisions, it is difficult to determine which decisions are responsible for the failure. There is neither a continuously available error signal nor a continuously available performance evaluation signal, as is the case in more conventional formulations of pole balancing. For example, Widrow and Smith (1964) used a linear regression method, implemented by an Adaline, to approximate the bang-bang control law required for balancing the pole. In order to use this method, however, they had to supply the controller with a signed error signal at each time step whose determination required external knowledge of the correct control decision for that time step. The problem we considered, on the other hand, requires the learning system to discover for itself which control decisions are correct, and in so doing, solve a difficult credit-assignment problem that is completely absent in the usual versions of this problem. If one were merely interested in controlling this type of dynamical system for practical purposes, one could easily obtain a more informative evaluation signal without resorting to an adaptive critic algorithm.* However, we think that the learning problem as posed here contains many of the difficulties that characterize the learning problem faced by adaptive elements that are embedded in networks.

Figure 33 shows a schematic representation of a cart to which a rigid pole is hinged. The cart is free to move within the bounds of a 1-dimensional track. The pole is free to move only in the vertical plane of the cart and track. The controller can apply an impulsive "left" or "right" force F of fixed magnitude to the cart at discrete time intervals. The cart-pole system was simulated so as to match as closely as possible the behavior of the real physical system, including nonlinearities and friction (see Barto, Sutton, and Anderson, 1983, for details). The cart-pole model has four state variables:

x: the position of the cart on the track,

θ: the angle of the pole with the vertical,

$\dot{x}$: the cart velocity, and

$\dot{\theta}$: the rate of change of the angle.

Parameters specify the pole length and mass, cart mass, coefficients of friction between the cart and the track and at the hinge between the pole and the cart, the impulsive control force magnitude, the force due to gravity, and the simulation time step size.

At each time step, the controller receives a vector giving the cart-pole system's state at that instant. If the pole falls or the cart hits the track boundary, the controller receives a failure signal, the cart-pole system (but not the controller's memory) is reset to its initial state, and another learning trial begins. The controller must attempt to generate controlling forces in order to avoid the failure signal for as long as possible.

We used the same method of representing the cart-pole state that Michie and Cham-

* In particular, since signals giving the state of the cart-pole system are available to the learning system, one could provide at each time step a signed error giving the difference between the current state and the desired state. The point is, however, that we assume that this desired state is unknown so that such an error signal is not available.

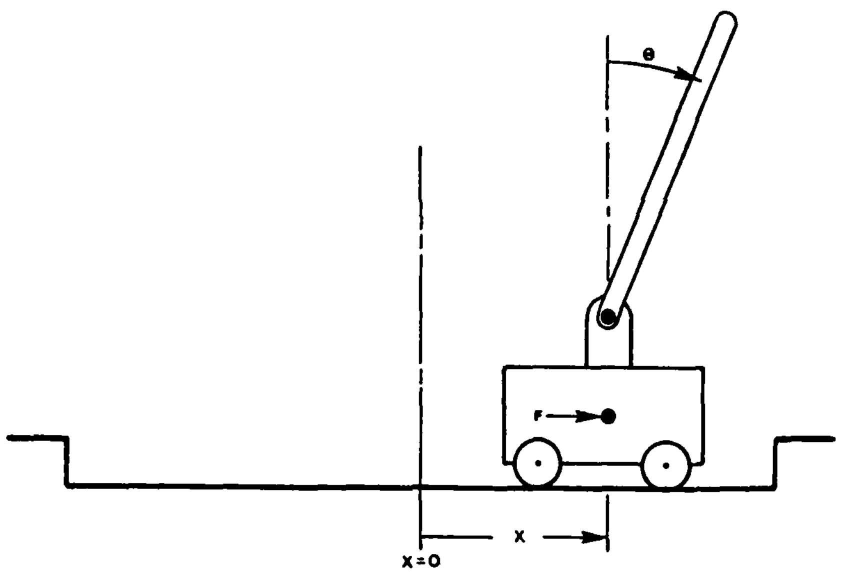


Figure 33

Cart-Pole System.

bers did. They divided the four-dimensional cart-pole state space into disjoint regions (or "boxes") by quantizing the four state variables. They distinguished 3 grades of cart position, 6 of pole angle, 3 of cart velocity, and 3 of pole anglular velocity. In our version of this task, the following quantization thresholds are used:

x: $\pm 0.8, \pm 2.4$ m,

θ: $0, \pm 1, \pm 6, \pm 12$ degrees,

$\dot{x}$: $\pm 0.5, \pm\infty$ m/sec,

$\dot{\theta}$: $\pm 50, \pm\infty$ deg/sec.

This yields $3 \times 3 \times 6 \times 3 = 162$ regions corresponding to all of the combinations of the intervals. We assume that this quantization is provided from the start. We assume the existence of a decoder that has four real-valued input pathways for the system state variables and 162 binary-valued output pathways corresponding to the 162 regions of the state space. The decoder transforms each state vector into a 162-component binary vector whose components are all zero except for a single 1 in the position corresponding to the region containing the state vector. This binary vector serves as input to an adaptive element that implements Algorithm 4 described in Section 4. We call this adaptive element an ***associative search element*** or ASE. The reinforcement signal sent to the ASE is generated by another element, which we call the ***adaptive critic element*** or ACE, that implements the AHC algorithm. The ACE receives as input the binary vectors generated by the decoder and the primary reinforcement signal. This reinforcement signal remains zero until failure occurs, at which time it is set to -1 for a single time step. Figure 34 shows an ASE together with an ACE configured for the pole-balancing task.

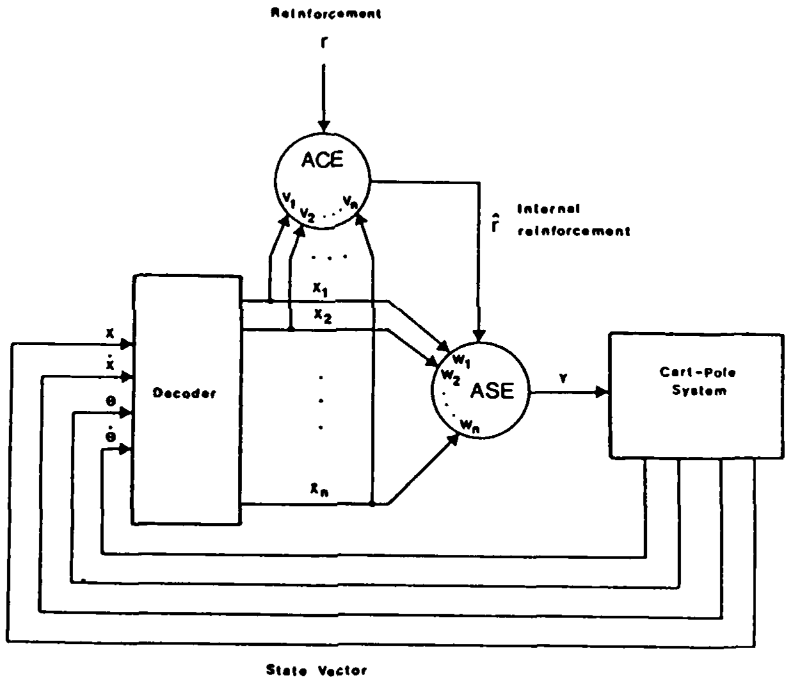


Figure 34

ASE and ACE Configured for the Pole-Balancing Task.

In the terminology introduced in Section 4, this two-element learning system uses an independent-associations approach to form both the mapping from cart-pole state vectors to control actions and the mapping determining heuristic renforcement as a function of cart-pole state. This allows direct comparison with Michie and Chambers' BOXES system. However, this is a restricted special case of the linear-mapping capabilities of the ASE and ACE. In future work we intend to explore other representations of cart-pole states that can permit the generalization abilities of a linear-mapping approach to be exploited. We also intend to replace the fixed decoder with a layered network whose task will be to create an appropriate representation for solving the problem (see Section 2).

We implemented the BOXES system as well as our system shown in Figure 34. We wanted to determine what kinds of neuronlike elements could attain or exceed the performance of the BOXES system. Our results suggest that a system using an ASE with heuristic reinforcement supplied by an ACE is easily able to out-perform the BOXES system. We must emphasize, however, that it is not our intention to criticize Michie and Chambers' program: The BOXES system they described was in an initial state of development and clearly can be extended to include a mechanism analogous to our ACE. We make comparisons with the performance of the BOXES system because it provides a convenient reference point.

We simulated a series of runs of each learning system attempting to control the same cart-pole simlation. Each run consisted of a sequence of trials, where each trial began with the cart-pole state $x = 0$, $\dot{x} = 0$, $\theta = 0$, $\dot{\theta} = 0$, and ended with a failure signal indicating that θ left the interval $[-12\,\text{degrees}, 12\,\text{degrees}]$ or x left the interval $[-2.4\,\text{m}, 2.4\,\text{m}]$. We also set all the trace variables to zero at the start of each trial. The learning systems were "naive" at the start of each run (i.e., all the weights w_i and v_i were set to zero). At the start of each run of the ASE/ACE system, we supplied a different seed to the pseudo-random number generator that we used to generate the randomness in the ACE's output. Since the ASE runs began with weight vectors equal to zero, initial actions for each state-space region were equiprobable, and initial ACE predictions were zero. Except for the random number generator seeds, identical parameter values were used for all runs. Runs consisted of 100 trials unless the run's duration exceeded 500,000 time steps (approximately 2.8 hours of simulated real time), in which case the run was terminated. For our implementation of the BOXES system, we used the parameter values published by Michie and Chambers (1968a). We experimented with other parameter values without obtaining consistently better performance. We did not attempt to optimize the performance of the systems using the ASE. We picked values that seemed reasonable based on our previous experience with similar adaptive elements (see Barto, Sutton, and Anderson, 1983, for details).

Figures 35 and 36 show the results of our simulations of BOXES and the ASE/ACE system. The graphs of Figure 35 are averages of performance over the 10 runs that produced the individual graphs shown in Figure 36. In both figures, a single point is plotted for each bin of 5 trials giving the number of time steps until failure averaged over those 5 trials. Almost all runs of the ASE/ACE system, and one run of the BOXES system, were terminated after 500,000 time steps before all 100 trials took place (those whose graphs terminate short of 100 trials in Figure 36). We terminated the simulation before failure on the last trials of these runs. To produce the averages for all 100 trials shown in Figure 35, we needed to make special provision for the interrupted runs. If the duration of the trial underway when the run was interrupted was less than the duration of the immediately preceding (and therefore complete) trial, then we assigned to fictitious remaining trials the duration of that preceding trial. Otherwise, we assigned to fictitious remaining trials the duration of the last trial when it was interrupted. We did this to prevent any short interrupted trials from producing deceptively low averages.

The ASE/ACE system achieved much longer runs than did the BOXES system. Figure 36 shows that the ACE/ASE system tended to solve the problem before it had experienced 100 failures, whereas the BOXES system tended not to. The good performance of the ASE/ACE system was almost entirely due to the ACE's supplying reinforcement throughout trials. For the BOXES system and for an ASE without an ACE, learning occurred only upon failure, an event that became less frequent as learning proceeded. With the ACE in place, an ASE could receive evaluative feedback on every time step. The learning produced by this feedback caused the system to attempt to enter particular parts of the state space and to avoid others. We simulated the control problem using an ASE without an ACE, using the same parameter settings that worked well for the ASE/ACE experiments. The ASE was not able to attain the level of performance shown by the BOXES system. However, we have not yet systematically evaluated and compared the performances of

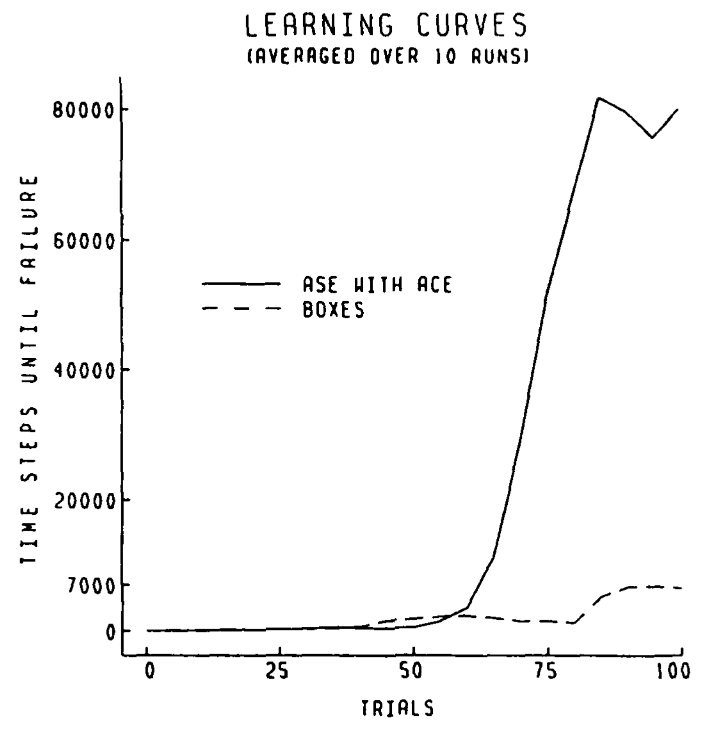


Figure 35

Averaged Learning Curves for ASE/ACE system and BOXES system on the Pole-Balancing Task.

other associative-reinforcement learning algorithms with and without the use of the AHC algorithm. We also have not applied the time-until-failure version of the AHC algorithm developed above.

Conclusion

We have attempted to justify an adaptive heuristic critic algorithm. The basic idea is to approximate an ideal reinforcement signal that would provide an immediate indication as to whether or not an action would lead to better than average future cumulative reinforcement when taken in the current situation. The resulting algorithm turns out to be similar to the "generalization learning" algorithm used for altering the evaluation function in Samuel's (1959) checkers playing program. If Samuel's algorithm is simplified by, among

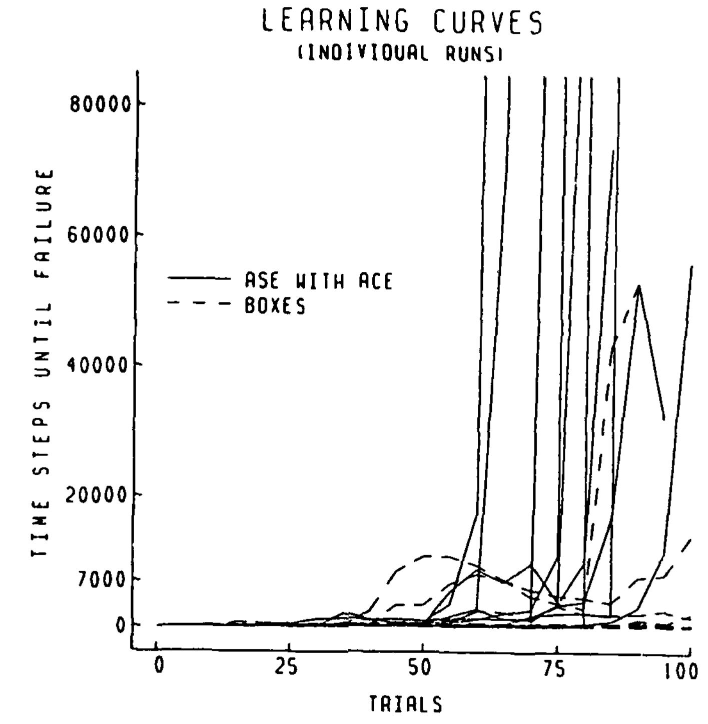


Figure 36

Learning Curves for Individual Runs of the ASE/ACE system and BOXES system on the Pole-Balancing Task.

other things, removing all features that are specialized for 1) the game of checkers, for 2) tasks in which off-line search is performed, and for 3) efficient implementation on a small computer, the only difference between what remains and the AHC algorithm is that the latter includes the discount rate parameter γ whereas Samuel's algorithm appears not to. Theoretical analysis in terms of the ideal reinforcement signal suggests that whereas the discount rate parameter is not needed for the special case of time-blind tasks, which includes game-playing tasks, it may be necessary for other tasks. Experiments with abstract tasks and with a more difficult pole-balancing task confirm this conclusion by showing greatly improved performance through the use of the discount rate parameter (Sutton, forthcoming).

Section 7
SUMMARY

In this report we described some of the results obtained from our simulation experiments with goal-seeking elements. The results presented in Section 2 show that goal-seeking elements are able to cooperate to solve problems that individual elements cannot solve. These experiments with layered networks indicate that the approach we are taking does have the potential for making adaptive networks useful substrates for adaptive problem solving. These experiments also indicate that more development would be useful at the level of basic algorithms for single elements. We gained increased understanding of the type of learning problem faced by elements that are embedded in networks and came to appreciate that rapid and reliable learning by networks requires that components implement learning algorithms that are effective in a wide range of conditions. Unlike learning systems that are presented with carefully controlled problems by knowledgeable teachers, network components face environments, determined mostly by the rest of the network, that do not present neatly structured problems. Components must be opportunistic agents capable of improving performance in a wide range of conditions and under considerable uncertainty. Failure of components under these conditions manifests itself as poor performance by the network. Therefore, to correctly attribute poor network performance to particular design decisions made at the *network* level, we felt the need to continue the development of individual components as detailed in Sections 3 through 6 of this report.

The learning problem faced by a network component can be characterized as an associative reinforcement learning problem under uncertainty as described in Section 1. Contrary to popular belief, this type of problem has not been extensively studied. We isolated and examined several problems that arise in this type of learning which might be regarded as aspects of the credit-assignment problem. By systematically experimenting with a set of algorithms over a wide range of tasks, we made progress in three areas: 1) we isolated and elucidated several issues concerning sequential credit assignment, including those produced by unbalanced reinforcement, misleading generalizations, delayed reinforcement, and secondary reinforcement, 2) we demonstrated that substantial performance improvements are possible in reinforcement-learning problems through the use of reinforcement-comparison mechanisms, and 3) we presented an algorithm, the AHC algorithm, that can make temporal credit-assignment problems less severe. We demonstrated the utility of this algorithm using the pole-balancing control problem.

Reinforcement Comparison — Reinforcement-comparison mechanisms played a prominent role in the research we reported. In Section 3 we described several reinforcement-comparison algorithms for discrete-action nonassociative reinforcement-learning tasks. We described results showing that these algorithms led to substantial increases in learning rate compared to several algorithms not employing reinforcement-comparison mechanisms, including some well-studied learning automaton algorithms. Their learning speed was particularly evident for tasks involving unbalanced reward probabilities. Although reinforcement comparison is a necessary part of any algorithm designed to perform local, gradient-directed search of a continuous parameter space, it is not at all obvious what utility such a comparison would have in algorithms designed for the global search of a set of unrelated, discrete actions. Our results suggest that reinforcement comparison is also useful for these types of problems.

Misleading Generalizations — The results described in Section 4 concern algorithms that are extensions to the associative case of the nonassociative algorithms discussed in Section 3. In these experiments, we provided neutral input in addition to reinforcement input to the learning system. However, whereas a learning system's actions could influence subsequent reinforcement, they could not influence subsequent neutral input. This permitted us to focus on some of the basic problems that arise when a system [illegible]st fo[illegible] a mapping from input patterns to actions under the influence of reinforcement feedl[illegible]. We found that even if the required mapping were within the representational capabilities of the learning system, it was possible for certain characteristics of the problem to prevent the formation of the correct mapping. This occurred as a result of a variety of asymmetries that favored the association of the correct action with some input patterns while at the same time prevented the formation of the correct association with other input patterns. Generalization from the favored pattern could overwhelm learning for the other pattern, preventing the necessary discrimination.

We consider this problem to be one of great importance. Generalization between input patterns has the potential for greatly accelerating learning, but brings with it the danger of preventing the formation of certain types of mappings. This is particularly troublesome given our interest in using elements capable of associative reinforcement learning as network components. First, it will not be possible to ensure the absence of reinforcement asymmetries in the tasks faced by network components because their environments will be largely determined by the rest of the network, which will itself be adapting. Second, since we wish to use networks as the means for overcoming the representational limitations of single elements, it seems necessary that the elements be able to form any mapping that is within their own representation capabilities. If the correct sort of "pressure" required for forcing even linear discrimination is not generated by the elements, it is hard to see how the appropriate kind of pressure can be generated in collections of elements for forcing the development of new features.

We are aware of no previous studies that recognized these types of problems. This is probably due to the sparsity of research on associative reinforcement learning using the linear-mapping approach. We experimented with an algorithm similar to one of the few relevant algorithms studied in the past. We found that it was very susceptible to the problems due to misleading generalizations. Under certain conditions, it always converged to the incorrect mapping. We described other algorithms, which incorporate generalizations of the reinforcement-comparison mechanisms described in Section 3, that can overcome these difficulties.

Delayed Reinforcement — Our results concerning delayed reinforcement, described in Section 5, confirm the usual criticism of the "recency heuristic" for assigning credit. For a backwards-averaging method of implementing the recency heuristic, the decrease in learning rate is directly proportional to the length of the time interval over which the average is taken, and an average that does not reach far enough back in time for a given reinforcement delay is useless for assigning credit. In selecting the interval for backwards averaging one must accept a tradeoff in performance on short-delay tasks and performance on long-delay tasks. Trace lengths optimal for long delays result in a suboptimal learning rate on tasks with short delays. We pointed out, however, that any *a priori* knowledge about reinforcement delay can be incorporated into the backwards-averaging kernel and that this kernel might be adaptively adjusted. We did not pursue either of these possibilities.

One of the more interesting results described in Section 5 is related to the results of Section 4 concerning misleading generalization. With backwards-averaging methods, it is possible for the learning due to a short-term reward to prevent learning due to a longer-term reward in situations in which the latter is clearly better in terms of cumulative reward. This is another phenomenon that illustrates the weakness of the recency heuristic. All the algorithms exclusively employing backwards-averaging for handling delayed reinforcement performed poorly on this type of task. They performed poorly not just because they were learning slowly, but because they were actually learning to perform the wrong action. Better performance would have resulted if no "learning" had occurred at all. Although algorithms employing a reinforcement-comparison mechanism did better on these tasks than the other algorithms, they still performed poorly.

Like the conditions that bring about incorrect learning due to misleading generalization, the conditions that bring about this deficiency can be expected to occur within networks. This is therefore another example of a low-level problem that can manifest itself in poor performance at the network level.

Secondary Reinforcement — In Section 6 we developed a secondary reinforcement algorithm, the AHC algorithm, that converts neutral stimuli to reinforcement signals in order to overcome some of the shortcomings of the recency heuristic for sequential credit assignment. The resulting algorithm can be regarded as an extension of the reinforcement-comparison algorithms described in earlier sections by the addition of a reinforcement-anticipation mechanism. In order for this algorithm to be useful, the learning system must be able to control neutral as well as reinforcement input.

We made use of the notion of an *ideal reinforcement signal.* This theoretical construction assisted in our understanding of existing sequential credit assignment algorithms and led to the creation of new ones. We illustrated the utility of the resulting AHC algorithm by using a particular formulation of the pole-balancing control task. Our learning system performed significantly better than did a learning system designed specifically for the pole-balancing task that did not include secondary reinforcement.

Centralised versus Distributed Algorithms — Aside from the results described in Section 2, which explicitly deal with networks, the results described in this report can be regarded as concerning either individual adaptive components of networks or as centralized learning algorithms that bear no relation to networks. If the first view is adopted, then these results are indirectly about distributed algorithms, implementable as adaptive networks. On the other hand, nothing prevents the latter view, and we believe many of our observations are also significant for this centralized interpretation. However, despite this possibility, we are really interested in distributed learning algorithms, and we believe that our research program represents a rather significant departure from the usual view of adaptive elements, adaptive networks, and related neural metaphors.

Our adaptive components implement simple input/output functions and communicate with one another by means of excitatory and inhibitory signals rather than by means of complex symbolic messages. They resemble most neuronlike adaptive elements in this regard. However, the algorithms we use to adjust the parameters of the input/output function are more complex than those generally considered in the past. They require numerous "auxiliary" variables to implement eligibility traces, reinforcement-comparison mechanisms, and reinforcement-anticipation mechanisms. We argued that these mechanisms are necessary if the element is to be capable of efficient learning in environments that do not provide controlled training experiences for the element. In environments such as these, the element needs to perform functions for itself that are more often assumed to be carried out by a helpful teacher. Irrespective of the precise manner of "packaging" these functions in terms of individual neuronlike elements, this perspective places considerable adaptive power at a low level in the functional/structural hierarchy.

In contrast to this approach are those that are completely centralized or that are distributed across networks of components that individually implement very sophisticated computational functions. Our research suggests that complex learning algorithms are required to adjust even very simple input/output functions in unhelpful environments. It therefore also suggests that *extremely* complex learning algorithms may be required to adjust the complex computations performed by centralized or coarsely distributed systems under similar conditions. We therefore think that an approach using compuationally simple elements that employ sophisticated learning algorithms is a more promising avenue for achieving deeply adaptive problem-solving systems.

REFERENCES

Alder, M. D. A convergence theorem for hierarchies of model neurones. *SIAM J. Comput.*, 1975, *4*, 192–201.

Anderson, C. W. Feature generation and selection by a layered network of reinforcement learning elements: Some initial experiments. COINS Technical Report 82-12, University of Massachusetts, Amherst, 1982.

Atkinson, R.C., Bower, G.H., & Crothers, E.J. it An Introduction to Mathematical Learning Theory. New York: Wiley, 1965.

Atkinson, R.C., & Estes, W.K. *Stimulus sampling theory.* In R.D. Luce, R.R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology, Vol. II.* New York: Wiley, 1963.

Ballard, D. H. Parameter networks: Toward a theory of low-level vision. *Proceedings 7th IJCAI,* Vancouver, B. C., August 1981.

Barr, A. & Feigenbaum, E. A. *The handbook of artificial intelligence, Volume 1.* Los Altos, California: Kauffman, 1981.

Barto, A. G., & Sutton, R. S. Goal seeking components for adaptive intelligence: An initial assessment. *Air Force Wright Aeronautical Laboratories/Avionics Laboratory Technical Report AFWAL-TR-81-1070,* Wright-Patterson AFB, Ohio, 1981a.

Barto, A. R., Anderson, C. W., & Sutton, R. S. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 1982, *43* 175–185.

Barto, A. G., & Sutton. R. S. Landmark learning: An illustration of associative search. *Bioliological Cybernetics*, 1981, *42*, 1–8.

Barto, A. G., & Sutton, R. S. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioral Brain Research*, 1982, *4*, 221–235.

Barto, A. G., Sutton, R. S., & Anderson, C. W. Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 1983.

Barto, A. G., Sutton, R. S., & Brouwer, P. S. Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 1981, *40*, 201–211.

Block, H. D., Nilsson, N. J., & Duda, R. O. Determination and detection of features in patterns. In Tou, J. T. & Wilcox, R. H. (Eds.), *Computer and information sciences: Collected papers in learning, adaptation, and control in information systems.* Washington, D. C.: Spartan Books, 1964, 75–110.

Bradt, R. N, Johnson, S. M., & Karlin, S. On sequential designs for maximizing the sum of n observations. *Ann. Math. Stat.*, 1956, *27*, 1060–1074.

Bush, R. R., & Estes, W. K. (Eds.). *Studies in mathematical learning theory.* Stanford: Stanford University Press, 1959.

Bush, R. R., & Mosteller, F. *Stochastic models for learning.* New York: Wiley, 1955.

Campbell, D. T. Blind variation and selective survival as a general strategy in knowledge-processes. In M. C. Yovits, S. Cameron (Eds.), *Self-organizing systems.* New York: Pergamon, 1962, 201–231.

Carterette, T. S. An application of stimulus sampling theory to summated generalization. *Journal of Experimental Psychology*, 1961, *62*, 448–455.

Cohen P. R. & Feigenbaum, E. A. *The handbook of artificial intelligence, Volume 3.* Los Altos, California: Kauffman, 1982.

Cover, T. M., & Hellman, M. E. The two-armed bandit problem with time-invariant finite memory. *IEEE Transactions on Information Theory 16*, 1970, 185–195.

Derman, C. *Finite state markovian decision processes.* New York: Academic Press, 1970.

Duda, R. O., & Hart, P. E. *Pattern classification and scene analysis.* New York: Wiley, 1973.

Edelman, G. M. Group selection and phasic reentrant signalling: A theory of higher brain function. In *The mindful brain: Cortical organization and the group-selective theory of higher brain function.* In G. M. Edelman & V. B. Mountcastle (Eds.), Cambridge, MA: The MIT press, 1978.

Estes, W.K. Toward a statistical theory of learning. *Psychololgical Review*, 1950, *57*, 94–107.

Farley, B. G., & Clark, W. A. Simulation of self-organizing systems by digital computer. *I.R.E. Transactions on Inf. Theory*, *4*, 1954, 76–84, 1954.

Feldman, J. A. A connectionist model of visual memory. In Hinton, G., & Anderson, J. (Eds.), *Parallel models of associative memory.* Hillsdale, N. J: Erlbaum, 1981.

Friedman, M. P., Trabasso, T., & Mosberg, L. Tests of a mixed model for paired-associates learning with overlapping stimuli. *Journal of Mathematical Psychology*, 1967, 4, 316–334.

Fukushima, K. A model of associative memory in the brain. *Kybernetic*, 1973, *12*, 58–63.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980, *36*, 193–202.

Geman, S. Stochastic relaxation methods for image restoration and expert systems. In D. B. Cooper, R. L. Launer, & D. E. McClure (Eds.), *Automated image analysis: Theory and experiments.* N.Y.: Academic Press, 1984.

Geman, D. & Geman, S. Parameter estimation for some Markov random fields. Brown University Division of Applied Mathematics Tech. Rept. August, 1983.

Gilstrap, L. O., Jr. Keys to developing machines with high-level artificial intelligence. *Design Engineering Conference, ASME,* New York, 1971.

Hampson, S., & Kibler, D. A boolean complete neural model of adaptive behavior. Department of Information & Computer Science Technical Report No. 190, University of California, Irvine, CA, 1982.

Hinton, G. E. Implementing semantic networks in parallel hardware. In Hinton, G. & Anderson, J. (Eds.), *Parallel models of associative memory.* Hilsdale, N. J.: Erlbaum 1981, 161–187

Hinton, G. E., & Sejnowski, T. J. Analyzing Cooperative Computation. *Proceedings of the Fifth Annual Conference of the Cognitive Science Society,* Rochester N.Y., 1983.

Holland, J. H. Adaptive algorithms for discovering and using general patterns in growing knowledge-bases. *International Journal of Policy Analysis and Information Systems,* 1980, *4,* 217–240.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.* USA, 1982, *79,* 2554–2558.

Ivanhenko, A. G. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics,* 1971, *1.*

Kehoe, E. J. Conditioning with serial compound stimuli: Theoretical and empirical issues. *Experimental Animal Behavior,* 1982, *1,* 30–65.

Klopf, A. H. & Gose, E. An evolutionary pattern recognition network. *IEEE Transactions on Systems, Science and Cybernetics,* 1969, *5,* 247–250.

A. H. Klopf. Brain function and adaptive systems– A heterostatic theory. *Air Force Cambridge Research Laboratories Research Report,* AFCRL-72-0164, Bedford, MA., 1972 (A summary appears in *Proceedings International Conference on Systems, Man, Cybernetics). IEEE Systems, Man, and Cybernetics Society,* 1974, Dallas, Texas.

Klopf, A. H. *The hedonistic neuron: A theory of memory, learning, and intelligence.* Washington, D.C.: Hemisphere, 1982.

La Berge, D. L. Generalization gradients in a discrimination situation. *Journal of Experimental Psychology*, 1961, *62*, 88–94.

Lakshmivarahan, S. *Learning Algorithms and Applications.* Springer-Verlag, 1981.

Lenat, D. B., Hayes-Roth, F., Klahr, P. Cognitive economy. Stanford Heuristic Programming Project HPP-79-15 (working paper), 1979.

Lovejoy, E. *Attention in discrimination learning.* San Francisco: Holden-Day, 1968.

Luce, R. D., Bush, R. R, & Galanter, E. (Eds.). *Handbook of mathematical psychology,* Vols. I, II. New York: Wiley, 1963.

Luce, R. D., Bush, R. R., & Galanter, E. (Eds.). *Handbook of mathematical psychology,* Vol. III. New York: Wiley, 1965.

Marrs, P. & Poppelbaum, E. J. *Stochastic and deterministic averaging processors.* London: The Institute of Electrical Engineers, 1981.

Mason, L.G. An optimal S-model learning algorithm for S-model learning environments. *IEEE Trans. on Automatic Control,* 1973, 493–496.

McClelland, J. L., & Rumelhart, D. E. An interactive model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review,* 1981, *88,* 275–407.

Mendel, J. M., & McLaren, R. W. Reinforcement learning control and pattern recognition systems. In Mendel, J. M., & Fu, K. S. (Eds.), *Adaptive, learning and pattern recognition systems: Theory and applications,* New York: Academic Press, 1970, 287–318.

Michie, D., & Chambers, R. A. BOXES: An experiment in adaptive control. In Dale, E., & Michie, D. (Eds.), *Machine Intelligence 2.* Edinburgh: Oliver and Boyd, 1968a, 137–152.

Michie, D., & Chambers, R. A. 'Boxes' as a model of pattern-formation. In Waddington, C. H. (Ed.), *Towards a theoretical biology; 1, Prolegomena.* Edinburgh: Edinburgh University Press, 1968b, 206–215.

Mine, H., & Osaki, S. *Markovian decision processes.* New York: American Elsevier Publishing Company, Inc., 1970.

Minsky, M. L. Theory of neural-analog reinforcement systems and its application to the brain-model problem. Princeton Univ. Ph.D. Dissertation. 1954.

Minsky, M. L. Steps toward artificial intelligence. *Proceedings IRE,* 1961, *49,* 8–30. (Reprinted in Feigenbaum, E. A., & Feldman, J. *Computers and thought.* New York: McGraw-Hill, 1963, 406–450.)

Minsky, M.L. & Selfridge, O.G. Learning in random nets. In C. Cherry (Ed.), *Information theory: Fourth London Symposium.* London: Butterworths, 1961.

Minsky, M. L., & Papert, S. *Perceptrons: An introduction to computational geometry.* Cambridge, MA: MIT Press, 1969.

Narendra, K. S., & Thathachar, M. A. L. Learning automata—a survey. *IEEE Transactions on Systems, Man, and Cybernetics,* 1974, *4,* 323–334.

Raibert, M. H. A model for sensorimotor control and learning. *Biological Cybernetics,* 1978 , *29,* 29–36.

Reilly, D. L., Cooper, L. N., & Elbaum, C. A neural model for category learning. *Biological Cybernetics,* 1982, *45,* 35–41.

Restle, F. A theory of discrimination learning. *Psychological Review,* 1955, *62,* 11–19.

Rosenblatt, F. *Principles of neurodynamics.* New York: Spartan Books, 1962.

Rumelhart, D. E., & McClelland, J. L. An interactive activation model of context effects in letter perception: Part 2. The contextual enhancement effect and some tests and extensions of the model. *Psychological Review,* 1982, *89,* 60–94.

Samuel, A.L. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development,* 1959, *3,* 210–229.

Selfridge, O. G. Pandemonium: A paradigm for learning. *Proceedings of the Symposium on the Mechanisation of Thought Processes.* Teddington, England: National Physical Laboratory, H.M. Stationary Office, London, 2 vols., 1959.

Stafford, R. A. Multi-layer learning networks. In Garvey, J. E. (Ed.), *Symposium on self-organizing systems.* Office of Naval Research, 1963.

Sutherland, N. S., & Mackintosh, N. J. *Mechanisms of animal discrimination learning.* New York: Academic Press, 1971.

Sutton, R. S. Temporal aspects of credit assignment in reinforcement learning. Univ. of Massachusetts Ph.D. Dissertation, forthcoming.

Sutton, R. S., & Barto, A. G. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review,* 1981, *88,* 135–171.

Thorndike, E. L. *Animal intelligence.* Darien, Conn.: Hafner, 1911.

Tsetlin, M. L. *Automaton theory and modelling of biological systems.* New York: Academic Press, 1973.

Widrow, B. Generalization and information storage in networks of adaline "neurons." In Yovits, M., Jacobi, G., & Goldstein, G. (Eds.), *Self-organizing systems.* Spartan Books, 1962.

Widrow, B., Gupta, N. K., & Maitra, S. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics,* 1973, *5,* 455–465.

Widrow B. & Hoff, M. E. Adaptive switching circuits. *1960 WESCON Convention Record Part IV,* 1960, 96–104.

Widrow, B., & Smith, F. W. Pattern-recognizing control systems. In Tow, J. T., & Wilcox, R. H.,*Computer and information sciences.* Clever Hume Press, 1964, 288–317.

Witten, I. H. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 1977, *34*, 286-295.

Zeaman, D., & House, B.J. The role of attention in retardate discrimination learning. In Ellis, N. R. (Ed.), *Handbook of mental deficiency.* New York: McGraw-Hill, 1963.